IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	: James CRAWFORD	Art Unit	: 2141
Serial No.	: 09/597,784	Examiner	: April Baugh
Filed	: June 19, 2000		
Title	: DIRECT FILE TRANSFER BETWEEN SUBSCRIBERS OF A COMMUNICATIONS SYSTEM		

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

DECLARATION UNDER 37 C.F.R. §1.131

I, James Crawford, hereby declare as follows:

1. I have read and understood the text of U.S. Application No. 09/597,784 (the '784 application), which discloses an invention for which I am the inventor.

2. On or prior to April 27, 2000, I reduced to practice the methods, computer programs, apparatus, and user interface described in paragraphs 4 and 5 of this document.

3. The attached pages are photocopies of:

a) a screen shot of a directory containing the AIM installer program (Exhibit 1) showing a date of March 1, 2000 associated with the program. The AIM installer program is used to install the Windows AIM version 3.5.1856 binary ("the AIM program").

b) a redacted source code listing of the portion of the AIM program used for file transfer functionality (Exhibit 2). Each line of code is numbered.

c) a redacted source code listing for the portion of the AIM program used to setup connections between the client and the host (Exhibit 3)

d) a screen shot of a user interface of the AIM program showing a file transfer window that enables a user to set file transfer preferences (Exhibit 4).

e) a screen shot of a user interface of the AIM program showing an instant messaging interface depicting a subscriber having a user identity "oscarlogan" selecting an option to get a file from another subscriber having a user identity "OscaRaina" (Exhibit 5).

f) a screen shot of a user interface of the AIM program showing a list of files that the user identity "oscarlogan" may attempt to get from the user identity "OscaRaina" (Exhibit 6).

g) a screen shot of a user interface of the AIM program showing a window that is presented to the user identity "oscarlogan" after selecting a file from the list of files displayed in Exhibit 6 (Exhibit 7).

h) a screen shot of a user interface of the AIM program showing a window that is presented to the user identity "oscarlogan" indicating the status of the file transfer (Exhibit 8).

i) a screen shot of a user interface of the AIM program showing a window that is presented to the user identity "oscarlogan" indicating a request from the user identity "OscaRaina" to get files from the disk directory belonging to the user identity "oscarlogan" (Exhibit 9).

4. With respect to independent claims 1, 14, 29-31, and 36 of the '784 application, I implemented and practiced a method, a computer program and an apparatus that transferred one or more files between clients.

Specifically, the following was implemented and practiced as evidenced by the source code listings of Exhibits 2 and 3:

(a) a connection was established with a communications system host

- Exhibit 3 –
    - Lines 2740-2773 SessSignOn
    - Lines 1346-1396 ConnCreate
    - Lines 1514-1577 ConnConnect
    - Lines 491-495 connDoServerLookup
    - Lines 171-208 connLookupHost
    - Lines 1048-1056 connWndProc
    - Lines 599-641 connEventLookupComplete
    - Lines 506-517 connDoServerConnect
    - Lines 248-262 connConnectToHost
    - Lines 1063-1080 connWndProc
    - Lines 2268-2306 connEventRecvReady

- Lines 2159-2265 connReceiveBlock
- Lines 4628-4747 connProcessFLAP
- Lines 4052-4060 connProcessSignOn
- Lines 739-743 connEventAwaitChallangeComplete
- Lines 569-592 connDoValidation
- Lines 3731-3873 connSendSignOn
- Lines 2394-2410 ConnSendPacket
- Lines 2333-2391 connEventSendReady

(b) a request to establish a direct connection was sent to or received from a client also connected to the communications system host.

- Exhibit 2 –
  - Lines 2031-2032 Process Menu command to Get File
  - Lines 1733-1767 DoStartGet
  - Lines 1705-1724 RequestAndListen

(c) when the client permitted establishment of the direct connection, a direct socket connection that bypasses the communications system host was established

- Exhibit 2 –
  - Lines 3300-3323 SockListen
  - Lines 3338-3373 SockAcceptReady
  - Lines 3375-3533 SockRecvReady

(d) and a transfer of one or more files from the client was initiated over the direct socket connection.

- Exhibit 2 -
  - Lines 3108-3131 File Listing Dialog
  - Lines 1492-1494 handle IDC_GET button from dialog
  - Lines 938-980 FTGetListItem
  - Lines 446-462 FTReInitHdr
  - Lines 3639-3654 SockSend
  - Lines 464-481 FtInitHdr

- Lines 3248-3827 SockXXX handle TCP i/o
- Lines 292-1158 FtXXX process bytes received from other client and save to disk

5. With respect to independent claim 45 of the '784 application, I implemented and practiced a user interface that controlled file transfers between clients.

Specifically, the following was implemented and practiced as evidenced by Exhibit 9:

(a) A first graphical user interface element is configured to notify an operator of a second client of a request from a first client to establish a direct connection to the second client. The request is communicated to the second client by a communications system host, and the direct connection bypasses the communications system host.

- Exhibit 9 –
  - The "Get File request from OscaRaina" window.

(b) A second graphical user interface element configured to enable an operator of the second client to authorize establishment of a direct connection and a file transfer over the direct connection.

- Exhibit 9 -
  - The "OK" button in the "Get File request from OscaRaina" window

6. The AIM program was produced or written by myself or under my direction on or prior to April 27, 2000, and the date from Exhibit 1 supports this fact. The screen shots of the user interfaces relating to the transfer of files between clients were produced using the AIM program.

7. The AIM program corresponds to the method, computer program, apparatus, and user interface described in paragraphs 4 and 5 of this document.

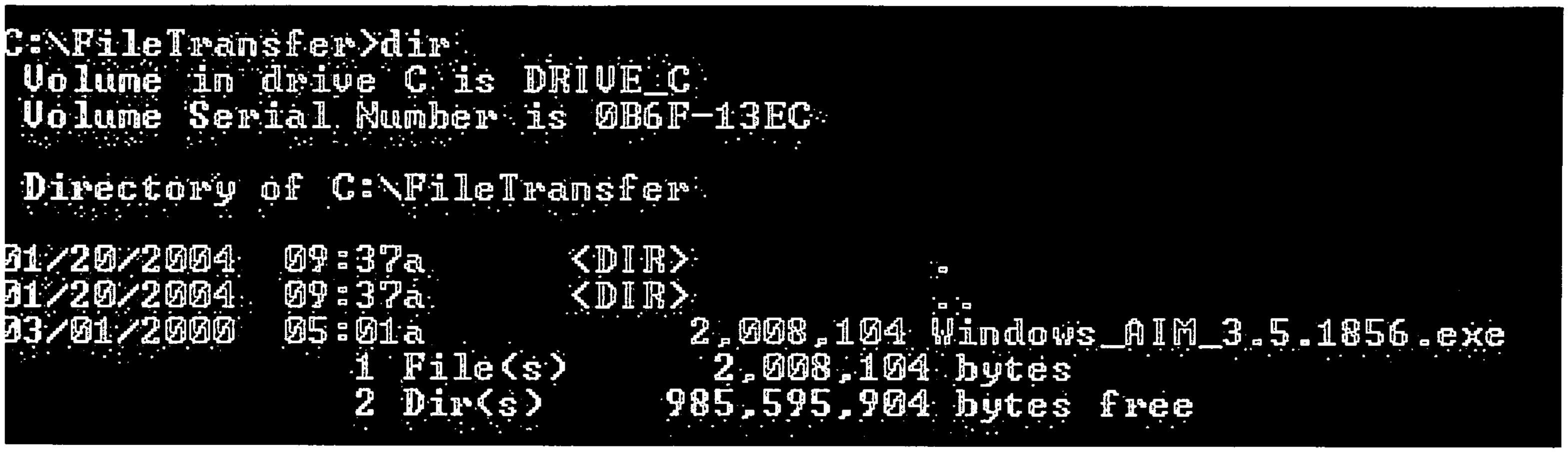
I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. §1001 and that such willful false statements may jeopardize the validity of the application or any patents issued thereon.

Signed and Declared at Belmont, MA this 6th day of ~~February~~ August, 2004

James Crawford

James Crawford

BEST AVAILABLE COPY



```
C:\FileTransfer>dir
 Volume in drive C is DRIVE_C
 Volume Serial Number is 0B6F-13EC

 Directory of C:\FileTransfer

01/20/2004  09:37a      <DIR>          .
01/20/2004  09:37a      <DIR>          ..
03/01/2000  05:01a           2,008,104 Windows_AIM_3.5.1856.exe
               1 File(s)      2,008,104 bytes
               2 Dir(s)     985,595,904 bytes free
```

Exhibit 1

```
void FTCloseFileAndSetTime(LPRENDEZVOUSTICKET rTicket)
{
    struct _utimbuf times;
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    FTHDR *shdr = &lpSrvStruct->sock_hdr;

    if (lpSrvStruct->fileP) {
        fclose(lpSrvStruct->fileP);
        lpSrvStruct->fileP = 0;

        times.actime = SWAP4(shdr->dwFiletime);
        times.modtime = times.actime;
        _utime(lpSrvStruct->dirPath,&times);
    }
}

DWORD FTCalcChecksum(FILE* fileP, DWORD fsize)
{
    WORD sum = 0;
```

Exhibit 2

```
    char buf[4096];
    while (fsize) {
        int n, nr = (fsize > sizeof(buf)) ? sizeof(buf) : fsize;
        n = fread(buf,1,nr,fileP);
        if (n != nr) {
            sum = 0;
            break;
        }
        sum = ComputeSum(sum, (LPWORD)buf, n);
        fsize -= n;
    }
    fseek(fileP, 0, SEEK_SET);
    sum = ~sum;
    return (DWORD)(0xffff & sum);
}

DWORD FTRecalcChecksum(DWORD chksum, LPWORD buf, long count)
{
    WORD sum = (WORD)(~chksum);
    sum = ComputeSum(sum,buf,count);
    sum = ~sum;
    return (DWORD)(0xffff & sum);
}

BOOL FTIsFileThere(LPSTR path, DWORD ftime, DWORD* chksumP, DWORD* fsizeP)
{
    FILE *fileP = fopen(path,"rb");
    *fsizeP = 0;
    *chksumP = 0;
    if (fileP) {
        struct _stat fst;
        _fstat(fileno(fileP), &fst);
        if (fst.st_mtime == (long)ftime) {
            *fsizeP = (DWORD)(filelength(fileno(fileP)));
            *chksumP = FTCalcChecksum(fileP,*fsizeP);
        }
        fclose(fileP);
        return TRUE;
    }
    return FALSE;
}

// this insures that there is a \ at the end
void FTSetDirPath(LPRENDEZVOUSTICKET rTicket, LPSTR in, BOOL buddyList)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    LPSTR cp, out = lpSrvStruct->dirPath;
    if (in)
        lstrcpy(out,in);
    cp  = _fstrrchr(out,'\\');
    if (cp) {
        cp++;
    } else {
        for (cp=out; *cp; cp++) ;
        *cp++ = '\\';
    }
    *cp = '\0';
    lpSrvStruct->dirPathOffset = cp;

    if (buddyList) {
        lstrcpy(cp,rTicket->nickname);
        while (*cp)  cp++;
        lstrcpy(cp,".lst");
    }
}

BOOL FTConstructDirListing(LPSTR filelib, DWORD *totSizeP, WORD *numFilesP)
{
    WIN32_FIND_DATA ffData;
    HANDLE ffh;
    FILE* fileP;
    LPSTR listName = LISTNAME;
    LPSTR logName = LOGNAME;
    LPSTR cp = filelib;
    while (*cp) cp++;

    *totSizeP = 0;
    *numFilesP = 0;
    lstrcpy(cp,listName);
    fileP = fopen(filelib,"w");
    if (!fileP) {
        return FALSE;
    }
    lstrcpy(cp,"*");
    ffh = FindFirstFile(filelib,&ffData);

    // TODO: how to tell path is a dir and not a * cmd?  isdir = TRUE;
    if (ffh != INVALID_HANDLE_VALUE) {
        do {
            if (ffData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
                // char dirpath[MAX_PATH];
                // TODO: go into subdirs also  (recursively call this func??)
            } else if (!lstrcmp(ffData.cFileName,listName) ||
                       !lstrcmp(ffData.cFileName,logName)) {
            } else if (!(ffData.dwFileAttributes & FILE_ATTRIBUTE_HIDDEN)) {
                char oneline[MAX_LIST_LINE];
                FILETIME locFiletime;
                FILETIME *ftimeP = &ffData.ftLastWriteTime;
                SYSTEMTIME systime;
                if (ftimeP->dwLowDateTime == 0 && ftimeP->dwHighDateTime == 0)
                    ftimeP = &ffData.ftCreationTime;
                FileTimeToLocalFileTime(ftimeP, &locFiletime);
                FileTimeToSystemTime(&locFiletime, &systime);
                sprintf(oneline,LINEFMT,
                        systime.wMonth,systime.wDay,systime.wYear,
                        systime.wHour,systime.wMinute,
                        ffData.nFileSizeLow, ffData.cFileName);
                fwrite(oneline,lstrlen(oneline),1,fileP);
                *numFilesP += 1;
                *totSizeP += ffData.nFileSizeLow;
            }
        } while (FindNextFile(ffh,&ffData));
        FindClose(ffh);
    }
    fclose(fileP);
    lstrcpy(cp,listName);
    return (FTSortFile(filelib,LINEOFF_NAME));
}

BOOL FTNextFile(LPSRVSTRUCT lpSrvStruct, LPSTR path)
{
    BOOL ret = TRUE;
    if (path) {
        lpSrvStruct->ffh = FindFirstFile(path,&lpSrvStruct->ffData);
        if (lpSrvStruct->ffh == INVALID_HANDLE_VALUE)
            return FALSE;
    } else
        ret = FindNextFile(lpSrvStruct->ffh,&lpSrvStruct->ffData);
    while (ret &&
            (lpSrvStruct->ffData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
        ret = FindNextFile(lpSrvStruct->ffh,&lpSrvStruct->ffData);
    return ret;
}


void FTReInitHdr(LPRENDEZVOUSTICKET rTicket)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    FTHDR *shdr = &lpSrvStruct->sock_hdr;

    lpSrvStruct->sock_starttime = GetTickCount();
    lpSrvStruct->sock_numSent = 0;
    lpSrvStruct->sock_numTotal = 0;
    lpSrvStruct->status_numTodo = 0;
    lpSrvStruct->totalNum = 1;
    lpSrvStruct->doneNum = 0;
    lpSrvStruct->status_lasttime = 0;
    lpSrvStruct->status_lastpercnt = 0;
    lpSrvStruct->totalSizeOfDoneFiles = 0;
    shdr->dwFilesize = 0;
    shdr->dwTotalFilesize = 0;
}

void FTInitHdr(LPRENDEZVOUSTICKET rTicket)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    FTHDR *shdr = &lpSrvStruct->sock_hdr;

    _fmemcpy(&shdr->bMagic[0], OSCAR_FT_MAGIC, sizeof(shdr->bMagic));
    shdr->wHdrType = StateToHdrType(lpSrvStruct->state);
    _fmemcpy(&shdr->bCookie[0], rTicket->cookie, sizeof(shdr->bCookie));
    shdr->wEncryption = 0;
    shdr->wCompression = 0;
    shdr->wTotalNumParts = SWAP2(1);
    shdr->wNumPartsLeft = SWAP2(1);
    shdr->dwTotalRessize = 0;
    shdr->dwRessize = 0;
    shdr->dwRestime = 0;
    shdr->dwResChecksum = 0;
    lstrcpy(&shdr->bIDstring[0],OSCAR_CLIENT_ID_STRING);
    _fmemset(&shdr->bDummy[0], 0, sizeof(shdr->bDummy));
    if ((lpSrvStruct->state == StateFileToSend ||
          lpSrvStruct->state == StateListToSend) &&
        lpSrvStruct->ffh != INVALID_HANDLE_VALUE) {
        WORD tmp;
        DWORD fsize, chksum;
        struct _stat fst;
        int hlen, bnamelen;
        FTCloseFileAndSetTime(rTicket);
        strncpy(&shdr->bName[0],lpSrvStruct->ffData.cFileName,FNSZ);
        bnamelen = lstrlen(lpSrvStruct->ffData.cFileName) + 1;
        if (bnamelen >= FNSZ) {
            shdr->bName[FNSZ-1] = '\0';
            bnamelen = FNSZ;
        }
        hlen = (sizeof(FTHDR) - FNSZ) + bnamelen;
        if (hlen < MIN_HDR_SZ)
            hlen = MIN_HDR_SZ;
        shdr->wHdrLen = SWAP2(hlen);
        lstrcpy(lpSrvStruct->dirPathOffset,lpSrvStruct->ffData.cFileName);
        lpSrvStruct->fileP = fopen(lpSrvStruct->dirPath,"rb");
        lpSrvStruct->sock_numSent = 0;
        lpSrvStruct->status_numTodo = 0;
        _fstat(fileno(lpSrvStruct->fileP), &fst);
        fsize = lpSrvStruct->ffData.nFileSizeLow;
        chksum = FTCalcChecksum(lpSrvStruct->fileP,fsize);
        shdr->dwTotalFilesize = SWAP4(lpSrvStruct->totalSize);
        shdr->wTotalNumFiles = SWAP2(lpSrvStruct->totalNum);
        tmp = lpSrvStruct->totalNum - lpSrvStruct->doneNum;
        shdr->wNumFilesLeft  = SWAP2(tmp);
        shdr->dwFilesize = SWAP4(fsize);
        shdr->dwFiletime = SWAP4(fst.st_mtime);
        shdr->dwChecksum = (chksum);
        shdr->dwNumRecvd = 0;
        shdr->dwRecvdChecksum = 0;
        shdr->bFlags = 0;
        if (lpSrvStruct->state == StateListToSend) {
            shdr->bListNameOffset = LINEOFF_NAME;
            shdr->bListSizeOffset = LINEOFF_SIZE;
            if (lpSrvStruct->sorted)
                shdr->bFlags |= FLAGS_SORTED;
            else
                shdr->bFlags &= ~FLAGS_SORTED;
        }
    }
}

// returns 0 for ok, 1 for err
BOOL FTValidateHdr(LPRENDEZVOUSTICKET rTicket)
{
    // insure it is a real header and one we are expecting; we dont want to
    // allow hacker client to change headers on us to do something unauthorized
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    FTHDR *shdr = &lpSrvStruct->sock_hdr;
    int hdrType = shdr->wHdrType;
    int state = lpSrvStruct->state;
    if (_fmemcmp(shdr->bMagic,OSCAR_FT_MAGIC,sizeof(shdr->bMagic)) ||
        _fmemcmp(shdr->bCookie, rTicket->cookie, sizeof(shdr->bCookie)) ||
        SWAP2(shdr->wHdrLen) < MIN_HDR_SZ ||
        !(hdrType & SR_MASK) ||
        // we shouldnt be recving a header we should be sending
        (lpSrvStruct->type & SR_MASK) == (hdrType & SR_MASK)) {
        return 1;
    }
    shdr->bIDstring[IDSZ-1]=0;
    shdr->bName[FNSZ-1]=0;
    return 0;
}

void ShowStatusWindow(LPRENDEZVOUSTICKET rTicket, int state)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (ISFT_SERVER(lpSrvStruct) &&
        ProfGetLong(PROF_USER, FT_KEY, FT_PUT_NO_STATUS))
        ShowWindow(rTicket->hDlgWnd, SW_HIDE);
    else
        ShowWindow(rTicket->hDlgWnd, state);
}

// returns 0 for ok, 1 for err, 2 for done
BOOL FTProcessHdr(LPRENDEZVOUSTICKET rTicket)
```

```
{
    TCHAR buf[512],buf1[512];
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    FTHDR *shdr = &lpSrvStruct->sock_hdr;
    int hdrType = shdr->wHdrType;
    int state = lpSrvStruct->state;
    BOOL sendhdr = -1;

    // Note this increments state to the next level
    if (hdrType == HDR_TYPE_FILE_TO_SEND) {
        if (state == StateFileToSend) {
            BOOL normal = TRUE;
            DWORD fsize, chksum, fsize1, chksum1;
            lpSrvStruct->totalNum = SWAP2(shdr->wTotalNumFiles);
            lpSrvStruct->totalSize = SWAP4(shdr->dwTotalFilesize);
            if (rTicket->cmdID != CMDID_SEND_FILE) {
                int nd,nf;
                FTSetDirPath(rTicket,0,0);
                nd = (int)(lpSrvStruct->dirPathOffset
                           - &lpSrvStruct->dirPath[0]);
                nf = lstrlen(shdr->bName) + 1;
                if (nd + nf > MAX_PATH)
                    nf = MAX_PATH - nd;
                strncpy(lpSrvStruct->dirPathOffset,shdr->bName,nf);
                lpSrvStruct->dirPath[MAX_PATH-1] = '\0';
                if (CheckForSecurityHoles(shdr->bName)) {
                    SET_RENDEZVOUS_DECLINE(rTicket);
                    return 1;
                }
            }
            FTCloseFileAndSetTime(rTicket);
            if (FTIsFileThere(lpSrvStruct->dirPath,SWAP4(shdr->dwFiletime),
                              &chksum,&fsize)) {
                UINT id = 0;
                // file already exists; decide if we need to resume it
                fsize1 = SWAP4(shdr->dwFilesize);
                chksum1 = (shdr->dwChecksum);
                if (chksum == 0) {
                    // file exists but wrong timestamp or 0 length
                    LoadString(lpOCMInfo->hModule,IDSFT_ErrFileAlreadyExists1,
                               buf,sizeof buf);
                    sprintf(buf1,buf,lpSrvStruct->dirPath);
                    if (lpSrvStruct->flags & (FLAGS_YES|FLAGS_NO) ||
                        MessageBox(rTicket->hDlgWnd, buf1, 0,
                                MB_OKCANCEL|MB_DEFBUTTON2|MB_ICONEXCLAMATION)
                         ==IDOK) {
                        remove(lpSrvStruct->dirPath);
                    } else {
                        shdr->bFlags |= FLAGS_DONTWANT;
                        id = IDSFTP_RecvrDoneDontWant;
                    }
                } else if (fsize1 == fsize && chksum1 == chksum) {
                    // dont bother with this file
                    shdr->dwNumRecvd = SWAP4(fsize);
                    shdr->dwRecvdChecksum = (chksum);
                    shdr->bFlags |= FLAGS_IDENTICAL;
                    id = IDSFTP_RecvrDoneNothing;
                } else if (fsize < fsize1) {
                    // local file is smaller; offer to resume
                    lpSrvStruct->state = StateFileWantToResume;
                    shdr->dwNumRecvd = SWAP4(fsize);
                    shdr->dwRecvdChecksum = (chksum);
                    normal = FALSE;
                }
                if (id != 0) {
                    lpSrvStruct->state = StateFileFooter;
                    if (rTicket->cmdID == CMDID_GET_LIST)
                        shdr->bFlags |= FLAGS_CONT;
                    AppendMsg(rTicket,id,FALSE,ISFT_RCVR(lpSrvStruct),TRUE);
                    lpSrvStruct->doneNum++;
                    lpSrvStruct->totalSizeOfDoneFiles += fsize1;
                    FTInitFileList(rTicket, FALSE);
                    SET_RENDEZVOUS_DONE(rTicket);
                    normal = FALSE;
                }
                lpSrvStruct->flags &= ~(FLAGS_NO|FLAGS_YES);
            }
            if (normal) {
                lpSrvStruct->state = StateFileOkToSend;
                lpSrvStruct->fileP = fopen(lpSrvStruct->dirPath,"wb");
                if (!lpSrvStruct->fileP) {
                    LoadString(lpOCMInfo->hModule,IDSFT_ErrCantOpenFile,
                               buf,sizeof buf);
                    MessageBox(rTicket->hDlgWnd, buf, 0, MB_OK);
                    return 1;
                }
            }
            sendhdr = TRUE;
        }
    } else if (hdrType == HDR_TYPE_FILE_OK_TO_SEND) {
        if (state == StateFileOkToSend) {
            lpSrvStruct->state = StateFileData;
            sendhdr = FALSE;
        }
    } else if (hdrType == HDR_TYPE_FILE_WANT_TO_RESUME) {
        if (state == StateFileOkToSend) {
            DWORD chksum;
            DWORD nrecvd = SWAP4(shdr->dwNumRecvd);
            DWORD chksumr = (shdr->dwRecvdChecksum);
            BOOL good = FALSE;
            FILE* fileP = fopen(lpSrvStruct->dirPath,"rb");
            if (fileP) {
                chksum = FTCalcChecksum(fileP,nrecvd);
                fclose(fileP);
                if (chksum == chksumr) {
                    lpSrvStruct->sock_numSent = nrecvd;
                    lpSrvStruct->sock_numTotal += lpSrvStruct->sock_numSent;
                    good = TRUE;
                }
            }
            if (!good) {
                shdr->dwRecvdChecksum = 0;
                shdr->dwNumRecvd = 0;
                nrecvd = 0;
            }
            lpSrvStruct->state = StateFileToResume;
            sendhdr = TRUE;
        }
    } else if (hdrType == HDR_TYPE_FILE_TO_RESUME) {
        if (state == StateFileToResume) {
            DWORD nrecvd = SWAP4(shdr->dwNumRecvd);
            lpSrvStruct->state = StateFileOkToResume;
            lpSrvStruct->fileP = fopen(lpSrvStruct->dirPath,
                                       nrecvd ? "ab" : "wb");
            if (!lpSrvStruct->fileP) {
                LoadString(lpOCMInfo->hModule,IDSFT_ErrCantOpenFile,
                        buf,sizeof buf);
                MessageBox(rTicket->hDlgWnd, buf, 0, MB_OK);
                return 1;
            }
            if (nrecvd) {
                lpSrvStruct->sock_numTotal += nrecvd;
            }
            sendhdr = TRUE;
        }
    } else if (hdrType == HDR_TYPE_FILE_OK_TO_RESUME) {
        if (state == StateFileOkToResume) {
            lpSrvStruct->state = StateFileData;
            sendhdr = FALSE;
        }
    } else if (hdrType == HDR_TYPE_FILE_FOOTER) {
        if (state == StateFileFooter ||
            state == StateFileOkToSend) {
            UINT id = IDSFTP_RecvrDone;
            if (shdr->bFlags & FLAGS_DONTWANT)
                id = IDSFTP_RecvrDoneDontWant;
            else if (shdr->bFlags & FLAGS_IDENTICAL)
                id = IDSFTP_RecvrDoneNothing;
            else if (shdr->dwChecksum &&
                     shdr->dwChecksum != shdr->dwRecvdChecksum)
                id = IDSFTP_RecvrDoneBadSum;
            AppendMsg(rTicket, id, FALSE, ISFT_RCVR(lpSrvStruct),TRUE);
            lpSrvStruct->doneNum++;
            lpSrvStruct->totalSizeOfDoneFiles += SWAP4(shdr->dwFilesize);
            if (lpSrvStruct->doneNum < lpSrvStruct->totalNum &&
                FTNextFile(lpSrvStruct, 0)) {
                lpSrvStruct->state = StateFileToSend;
                sendhdr = TRUE;
            } else if (shdr->bFlags & FLAGS_CONT) {
                ShowStatusWindow(rTicket, SW_HIDE);
                lpSrvStruct->state = StateListWantToGet;
                lpSrvStruct->sock_flags |= SockReadyToReceiveHdr;
                SET_RENDEZVOUS_DONE(rTicket);
                SockStartWaitTimer(rTicket);
                return 0;
            } else {
                SET_RENDEZVOUS_DONE(rTicket);
                return 1;
            }
        }
    } else if (hdrType == HDR_TYPE_LIST_TO_SEND) {
        if (state == StateListToSend) {
            lpSrvStruct->totalNum = SWAP2(shdr->wTotalNumFiles);
            lpSrvStruct->totalSize = SWAP4(shdr->dwTotalFilesize);
            if (rTicket->cmdID == CMDID_GET_LIST) {
                FTSetDirPath(rTicket,0,TRUE);
                if (lpSrvStruct->fileP)
                    fclose(lpSrvStruct->fileP);
                lpSrvStruct->fileP = fopen(lpSrvStruct->dirPath,"wb");
                if (!lpSrvStruct->fileP) {
                    LoadString(lpOCMInfo->hModule,IDSFT_ErrCantOpenFile,
                               buf,sizeof buf);
                    MessageBox(rTicket->hDlgWnd, buf, 0, MB_OK);
                    return 1;
                }
                lpSrvStruct->state = StateListOkToSend;
                sendhdr = TRUE;
            }
        }
    } else if (hdrType == HDR_TYPE_LIST_OK_TO_SEND) {
        if (state == StateListOkToSend) {
            lpSrvStruct->state = StateListData;
            sendhdr = FALSE;
        }
    } else if (hdrType == HDR_TYPE_LIST_FOOTER) {
        if (state == StateListFooter) {
            AppendMsg(rTicket,IDSFTP_RecvrDone,FALSE,
                      ISFT_RCVR(lpSrvStruct),TRUE);
            SET_RENDEZVOUS_DONE(rTicket);
            if (shdr->bFlags & FLAGS_CONT) {
                ShowStatusWindow(rTicket, SW_HIDE);
                lpSrvStruct->state = StateListWantToGet;
                lpSrvStruct->sock_flags |= SockReadyToReceiveHdr;
                SockStartWaitTimer(rTicket);
                return 0;
            }
        }
    } else if (hdrType == HDR_TYPE_LIST_WANT_TO_GET) {
        if (state == StateListWantToGet) {
            FTReInitHdr(rTicket);
            lpSrvStruct->state = StateFileToSend;
            ShowStatusWindow(rTicket, SW_SHOW);
            FTPrepareForPut(rTicket, shdr->bName);
            sendhdr = TRUE;
            lpSrvStruct->sock_timeout = 0;
        }
    }
    if (sendhdr == -1)
        return 1;

    SockSend(rTicket,sendhdr);
    return 0;
}

void FTIncrementState(LPRENDEZVOUSTICKET rTicket)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    FTHDR *shdr = &lpSrvStruct->sock_hdr;
    int state = lpSrvStruct->state;

// There are 3 state paths that are followed;
// 1) FToSend->FOkToSend->FData->FFooter
// 2) FToSend->FWantToResume->FToResume->FOkToResume->FData->FFooter
// 3) LToSend->LOkToSend->LData->LFooter/LWantToGet->FToSend...
    switch (state) {
        case StateFileToSend:
          lpSrvStruct->state = StateFileOkToSend;
          break;
        case StateFileOkToSend:
          lpSrvStruct->state = StateFileData;
          break;
        case StateFileData:
          lpSrvStruct->state = StateFileFooter;
          if (rTicket->cmdID == CMDID_GET_LIST)
              shdr->bFlags |= FLAGS_CONT;
          break;
        case StateFileFooter:
          if (rTicket->cmdID == CMDID_GET_LIST)
              lpSrvStruct->state = StateListWantToGet;
          else
              lpSrvStruct->state = StateFileToSend;
          break;
        case StateFileWantToResume:
          lpSrvStruct->state = StateFileToResume;
          break;
        case StateFileToResume:
          lpSrvStruct->state = StateFileOkToResume;
          break;
        case StateFileOkToResume:
          lpSrvStruct->state = StateFileData;
          break;
        case StateListToSend:
          lpSrvStruct->state = StateListOkToSend;
          break;
        case StateListOkToSend:
          lpSrvStruct->state = StateListData;
          break;
        case StateListData:
          lpSrvStruct->state = StateListFooter;
          shdr->bFlags |= FLAGS_CONT;
          break;
        case StateListFooter:
          lpSrvStruct->state = StateListWantToGet;
          break;
        case StateListWantToGet:
          lpSrvStruct->state = StateFileToSend;
          break;
    }
}

BOOL FTCountFilesToSend(LPSTR path, DWORD* totsize, WORD* totnum)
{
    WIN32_FIND_DATA ffData;
    HANDLE ffh = FindFirstFile(path,&ffData);
    BOOL isdir = FALSE;

    // TODO: how to tell path is a dir and not a * cmd?  isdir = TRUE;
    *totnum = 0;
    *totsize = 0;
    if (ffh != INVALID_HANDLE_VALUE) {
      do {
        if (ffData.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY) {
            // char dirpath[MAX_PATH];
            // TODO: go into subdirs also  (recursively call this func)
        } else {
            *totnum += 1;
            *totsize += ffData.nFileSizeLow;
        }
      } while (FindNextFile(ffh,&ffData));
      FindClose(ffh);
    }
    return isdir;
}

void FTMakeLocalPath(LPRENDEZVOUSTICKET rTicket, LPSTR localpath, int size)
{
    LPSRVPROFT lpSrvProFt = (LPSRVPROFT)(rTicket->lpSrvProposal);
    GetDirDownload(localpath);
    MakeDir(localpath);
    if (rTicket->cmdID == CMDID_SEND_FILE) {
        LPSTR path = &lpSrvProFt->bName[0];
        LPSTR lp, lpe, cp = _fstrrchr(path,'\\');
        if (!cp)
            cp = _fstrrchr(path,'/');   // in case its Unix
        if (!cp)
```

```
            cp = _fstrrchr(path,':');    // in case its Mac
        if (!cp)
            cp = path;
        else
            cp++;
        for (lp=localpath; *lp; lp++);
        if (*(lp-1) != '\\')
            *lp++ = '\\';
        for (lpe=localpath+size-1; *cp && lp < lpe; )
            *lp++ = *cp++;
        *lp = '\0';
    }
}

BOOL FTPrepareForPut(LPRENDEZVOUSTICKET rTicket, LPSTR fp)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    BOOL isDir;
    WORD numFiles;
    DWORD totSize;
    char filelib[2*MAX_PATH];

    GetDirFilelib(filelib);

    if (*fp == '\0') {
        lpSrvStruct->state = StateListToSend;
        lpSrvStruct->sorted = FTConstructDirListing(filelib, &totSize,
                                                    &numFiles);
    } else if (CheckForSecurityHoles(fp)) {
        AppendMsg(rTicket, IDSFTP_RecvrDecline, FALSE, FALSE, FALSE);
        SET_RENDEZVOUS_DECLINE(rTicket);
        CleanUp(rTicket, 0);
        return FALSE;
    } else {
        LPSTR cp = filelib + lstrlen(filelib);
        lstrcpy(cp,fp);
        lpSrvStruct->state = StateFileToSend;
        isDir = FTCountFilesToSend(filelib, &totSize, &numFiles);
    }
    // if there are no files in the FILE_LIBRARY, return an IGNORE NAK
    if (!numFiles) {
        AppendMsg(rTicket, IDSFTP_HasNoFiles, FALSE, TRUE, FALSE);
        SET_RENDEZVOUS_IGNORE(rTicket);
        CleanUp(rTicket, 0);
        return FALSE;
    }
    lpSrvStruct->totalNum = numFiles;
    lpSrvStruct->totalSize = totSize;
    FTNextFile(lpSrvStruct, filelib);
    FTSetDirPath(rTicket,filelib,0);
    return TRUE;
}

void FTEnableGetBut(HWND hwndList, HWND hwndGet)
{
    //int selid = (int)SendMessage(hwndList, LB_GETCURSEL,0,0);
    //EnableWindow(hwndGet, selid != LB_ERR);
    int n = (int)SendMessage(hwndList, LB_GETSELCOUNT,0,0);
    EnableWindow(hwndGet, (n > 0));
}

void FTGetListItem(LPRENDEZVOUSTICKET rTicket)
{
    char oneline[MAX_LIST_LINE];
    LPSTR cp;
    DWORD num;
    int selid,n;
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    FTHDR *shdr = &lpSrvStruct->sock_hdr;
    HWND hwndDlg  = rTicket->hDlgWnd;
    HWND hwndList = GetDlgItem(hwndDlg, IDC_DIR_FILELIST);
    HWND hwndGet  = GetDlgItem(hwndDlg, IDC_GET);
    HWND hwndStop = GetDlgItem(hwndDlg, IDABORT);

    //selid = (int)SendMessage(hwndList, LB_GETCURSEL,0,0);
    //if (selid != LB_ERR) {
    n = (int)SendMessage(hwndList, LB_GETSELCOUNT,0,0);
    if (n > 0) {
        SendMessage(hwndList,LB_GETSELITEMS,1,(LPARAM)(LPSTR)&selid);
        lpSrvStruct->selID = selid;
        SendMessage(hwndList,LB_GETTEXT, selid, (LPARAM)(LPSTR)oneline);
        cp = oneline + shdr->bListNameOffset + 1; // one for extra " " at
begin
        lstrcpy(&shdr->bName[0],cp);
        FTReInitHdr(rTicket);
        cp = oneline + shdr->bListSizeOffset + 1; // one for extra " " at
begin
        num = (DWORD)atol(cp);
        shdr->dwFilesize = SWAP4(num);
        shdr->dwTotalFilesize = shdr->dwFilesize;
        lpSrvStruct->sock_timeout = 0;
        SockSend(rTicket,TRUE);

        ShowThermo(rTicket->hDlgWnd,SW_SHOW,SW_HIDE);
        SetFocus(hwndList);
        EnableWindow(hwndGet, FALSE);
        EnableWindow(hwndStop, TRUE);
        EnableWindow(hwndList, FALSE);
    } else {
        lpSrvStruct->totalSizeOfDoneFiles = 0;
        FTEnableGetBut(hwndList,hwndGet);
        EnableWindow(hwndStop, FALSE);
        EnableWindow(hwndList, TRUE);
        SockStartWaitTimer(rTicket);
    }
}

// returns if file is successfully sorted
BOOL FTSortFile(LPSTR path, WORD offset)
{
    BOOL ret = FALSE;
    // has to be "rb" in order for filelength to work right
    FILE *fileP = fopen(path,"rb");;
    if (fileP) {
        LPSTR fileMem,lp,lpe;
        LPSTR* names;
        LPSTR* np;
        LPSTR* np1;
        LPSTR* np2;
        int n, nn, nl = 0;
        long ntot, nr, size;

        size = filelength(fileno(fileP));
        if (!size)
            return TRUE;
        fileMem = (LPSTR)MemAlloc(size + size); // make room for ptrs
        lp = fileMem;
        lpe = fileMem + size;
        names = (LPSTR*)(lpe+2);
        np = names;
        ntot = size;
        *lpe = '\n';
        *(lpe+1) = 0;
        if (!fileMem) {
            fclose(fileP);
            return FALSE;
        }
        while (ntot) {
            nr = ntot;
            if (nr > 0x7fff)
                nr = 0x7fff;
            if (fread(lp,1,nr,fileP) == 0)
                break;
            lp += nr;
            ntot -= nr;
        }
        fclose(fileP);
        // fill in pointers to beginning of each row
        lp = fileMem;
        while (lp < lpe) {
            *np++ = lp;
            nl++;
            while (*lp != '\r' && *lp != '\n')
                lp++;
            while (*lp == '\r' || *lp == '\n')
                *lp++ = 0;
        }
        // now sort pointers; dont worry too much about speed
        for (n=nl; n; n--) {
            np1 = names;
            for (nn=1; nn<n; nn++) {
                LPSTR cp1,cp2;
                np2 = np1 + 1;
                cp1 = *np1 + offset;
                cp2 = *np2 + offset;
                if (lstrcmp(cp1,cp2) > 0) {
                    cp1 = *np1;
                    *np1 = *np2;
                    *np2 = cp1;
                }
                np1++;
            }
        }
        // now write the sorted lines back to disk
        fileP = fopen(path,"wb");
        if (fileP) {
            np1 = names;
            while (nl--) {
                int len = lstrlen(*np1);
                fwrite(*np1,len,1,fileP);
                fwrite("\r\n",2,1,fileP);
                np1++;
            }
            fclose(fileP);
            ret = TRUE;
        }
        MemFree(fileMem);
    }
    return ret;
```

```
}

void FTInitUnsorted(LPSTR path, HWND hwnd, HWND hwndList, int charwid)
{
    FILE *fileP;
    char oneline[MAX_LIST_LINE];
    RECT wndRect;
    int wold, w, wdel;

    SendMessage(hwndList, LB_RESETCONTENT, 0, 0);
    fileP = fopen(path,"rb");
    if (fileP) {
        int nr, npos = 0, selid = 0, maxn = 0, len;
        oneline[0] = ' ';   // replaced with * when file transferred
        while ((nr = fread(&oneline[1],1,sizeof(oneline)-2,fileP)) > 0) {
            int nn = 0;
            LPSTR line1 = &oneline[1];

            while (nn < nr && *line1 != '\r' && *line1 != '\n') {
                line1++;
                nn++;
            }
            while (*line1 == '\r' || *line1 == '\n') {
                *line1++ = '\0';
                nn++;
            }
            npos += nn;
            fseek(fileP, npos,SEEK_SET);

            SendMessage(hwndList,LB_INSERTSTRING,selid++,(LPARAM)&oneline[0]);
            len = lstrlen(oneline);
            if (len > maxn)
                maxn = len;
        }
        fclose(fileP);

        // expand the window to fit more chars
        GetWindowRect(hwndList, &wndRect);
        ScreenRectToClient(hwnd, &wndRect);
        wold = wndRect.right - wndRect.left;
        maxn += 4;  // to allow for scroll bar
        w = maxn * charwid;
        if (w > wold) {
            wdel = w - wold;
            MoveWindow(hwndList, wndRect.left, wndRect.top,
                       w, wndRect.bottom - wndRect.top, TRUE);
            GetWindowRect(hwnd, &wndRect);
            wold = wndRect.right - wndRect.left;
            MoveWindow(hwnd, wndRect.left, wndRect.top, wold + wdel,
                       wndRect.bottom - wndRect.top, TRUE);
        }
    }
}

void FTInitFileList(LPRENDEZVOUSTICKET rTicket, BOOL firstTime)
{
    FILE *fileP = NULL;
    char oneline[MAX_LIST_LINE], buf[256];
    HWND hwndDlg = rTicket->hDlgWnd;
    HWND hwndList = GetDlgItem(hwndDlg, IDC_DIR_FILELIST);
    HWND hwndGet  = GetDlgItem(hwndDlg, IDC_GET);
    HWND hwndStop = GetDlgItem(hwndDlg, IDABORT);
    HWND hwndText = GetDlgItem(hwndDlg, IDC_STATUS_TEXT1);
    HWND hwndTherm= GetDlgItem(hwndDlg, IDC_THERMO);
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    FTHDR *shdr = &lpSrvStruct->sock_hdr;
    int selid = 0;

    if (firstTime) {
        lpSrvStruct->totalSizeOfDoneFiles = 0;
        FTSetDirPath(rTicket,0,TRUE);
        if (!(shdr->bFlags & FLAGS_SORTED))
            FTSortFile(lpSrvStruct->dirPath, shdr->bListNameOffset);
        FTInitUnsorted(lpSrvStruct->dirPath, hwndDlg, hwndList,
                       lpSrvStruct->listFontWidth);

        LoadString(lpOCMInfo->hModule, IDSFT_FileListInfo, buf, sizeof buf);
        SetWindowText(hwndText, buf);
        ShowThermo(hwndDlg,SW_HIDE,SW_SHOW);
        EnableWindow(hwndGet,FALSE);
        EnableWindow(hwndStop,FALSE);
        SetFocus(hwndList);
    } else {
        selid = lpSrvStruct->selID;
        InvalidateRect(hwndDlg,0,TRUE);
        // we just finished transferring the selection; update the list box
        SendMessage(hwndList, LB_SETSEL, 0, selid);
        SendMessage(hwndList, LB_GETTEXT, selid, (LPARAM)(LPSTR)oneline);
        SendMessage(hwndList, LB_DELETESTRING, selid, 0);
        oneline[0] = '*';
        SendMessage(hwndList, LB_INSERTSTRING, selid, (LPARAM)oneline);
        FTGetListItem(rTicket);
        return;
    }
}
```

```
        case IDC_GET:
            FTGetListItem(rTicket);
            return 1;
```

```
void RequestAndListen(LPRENDEZVOUSTICKET rTicket, int ids, TCHAR* path)
{
    TCHAR preBuf[MAX_PRETEXT_SIZE], buf[512];

    LoadString(lpOCMInfo->hModule, ids, buf, sizeof(buf));
    wsprintf(preBuf, buf, path);
    rTicket->preText = preBuf;
    rTicket->timeoutTime = GetTickCount() + RENDEZVOUS_TIMEOUT_DEFAULT;
    SET_RENDEZVOUS_IPADDR(rTicket);

    if (OMSendMessageByRef(OMTYPE_REQUEST, OMGROUP_RENDEZVOUS,
                           OMSG_RENDEZVOUS_REQ_PROPOSAL,
                           sizeof(RENDEZVOUSTICKET), rTicket)) {
        HWND hwndDlg = rTicket->hDlgWnd;
        rTicket->hDlgWnd = 0;
        DestroyWindow(hwndDlg);
        OpenStatus(rTicket,TRUE);
        SockListen(rTicket);
    }
}

void DoSockConnect(LPRENDEZVOUSTICKET rTicket)
{
    OpenStatus(rTicket,FALSE);
    SockCleanup(rTicket);
    SockConnect(rTicket);
}

void DoStartGet(LPRENDEZVOUSTICKET rTicket, LPSTR path)
{
    HWND hwndDlg = rTicket->hDlgWnd;
    FILE *InStream = NULL;
    HWND hwndFile = GetDlgItem(hwndDlg, IDC_FILE);
    int len, ids;
    LPSRVPROFT lpSrvProFt;
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;

    len = lstrlen(path) + sizeof(SRVPROFT);
    lpSrvProFt = (LPSRVPROFT)rTicket->lpSrvProposal;
    if (!lpSrvProFt) {
        CleanUp(rTicket, 0);
        return;
    }
    rTicket->lenSrvProposal = len;
    len -= 4;
    lpSrvStruct->type = TypeGet;
    if (path[0] == '\0') {
        lpSrvStruct->state = StateListToSend;
        lpSrvProFt->wSubtype = SUBTYPE_GET_LIST;
        rTicket->cmdID = CMDID_GET_LIST;
        ids = IDSFTP_GetListRequest;
    } else {
        lpSrvStruct->state = StateFileToSend;
        lpSrvProFt->wSubtype = SUBTYPE_GET_FILES;
        ids = IDSFTP_GetterRequest;
    }
    lstrcpy(&lpSrvProFt->bName[0], path);
    lpSrvProFt->wTag = SWAP2(RENDEZVOUS_TLV_TAGS_SERVICE_DATA);
    lpSrvProFt->wLen = SWAP2(len);
    FTMakeLocalPath(rTicket, lpSrvStruct->dirPath,
                    sizeof(lpSrvStruct->dirPath));
    RequestAndListen(rTicket, ids, path);
}
```

```
if (rTicket->cmdID == CMDID_GET_LIST) {
    DoStartGet(rTicket, "");
```

```
BOOL DoStartPut(LPRENDEZVOUSTICKET rTicket)
{
    TCHAR buf[512];
    LPSTR fp;
    int ids;
    LPSRVPROFT lpSrvProFt = (LPSRVPROFT)(rTicket->lpSrvProposal);
    if (!lpSrvProFt ||
        (lpSrvProFt->wTag != SWAP2(RENDEZVOUS_TLV_TAGS_SERVICE_DATA))) {
        SET_RENDEZVOUS_BUSTED(rTicket);
        return FALSE;
    }

    SET_RENDEZVOUS_NO_PROMPT(rTicket);
    fp = &lpSrvProFt->bName[0];
    if (*fp == '\0')
        ids = IDSFTP_GetListRequest;
    else
        ids = IDSFTP_GetterRequest;

    LoadString(lpOCMInfo->hModule, ids, buf, sizeof(buf));
    wsprintf(rTicket->preText, buf, fp);
    rTicket->lenSrvStruct = sizeof(SRVSTRUCT);
    return TRUE;
}

```

```
void DoStartPutPost(LPRENDEZVOUSTICKET rTicket)
{
    // Note: the lpSrvStruct cannot be used before this
    long allow;
    LPSRVSTRUCT lpSrvStruct;
    LPSRVPROFT lpSrvProFt = (LPSRVPROFT)(rTicket->lpSrvProposal);

    lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    lpSrvStruct->ffh = INVALID_HANDLE_VALUE;

    allow = ProfGetLong(PROF_USER, FT_KEY, FT_GET_ALLOW);
    if (allow == AllowNoOne ||
        (allow == AllowBuddy && BuddyNotOnBuddyList(rTicket->nickname))) {
        SET_RENDEZVOUS_DECLINE(rTicket);
        CleanUp(rTicket, 0);
    } else {
        lpSrvStruct->type = TypePut;
        if (FTPrepareForPut(rTicket,&lpSrvProFt->bName[0]))
            DoSockConnect(rTicket);
    }
}
```

```
    else if (getCmdActivated && CMDID_IS_GET(rTicket->cmdID))
        return (DoStartPut(rTicket));
        if (CMDID_IS_GET(rTicket->cmdID))
            DoStartPutPost(rTicket):
```

```
        {
            LPRENDEZVOUSTICKET rTicket = (LPRENDEZVOUSTICKET)lpData;
            if (rTicket->hModule == lpOCMInfo->hModule) {
                LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
                if (!lpSrvStruct || !(lpSrvStruct->sock_flags & SockConnected)
||
                    rTicket->reason == OM_PROTO_USER_EXIT) {
                    CleanUp(rTicket,0);
                } else {
                    char buf[512];
                    if (!goingOffline) {
                        HWND hWndFocus = GetFocus();
                        HWND hWnd = GetTopmostAppWindow();
                        LoadString(lpOCMInfo->hModule,IDSFT_Offline,
                                   buf,sizeof buf);
                        DisableAppWindows();
                        goingOffline = MessageBox(hWnd, buf, 0,

MB_OKCANCEL|MB_DEFBUTTON1|MB_ICONEXCLAMATION);
                        EnableAppWindows();
                        SetFocus(hWndFocus);

// when Sesame is finally used...
//                          goingOffline=OkCancelBox(lpOCMInfo->hModule,
//                                                  IDSFT_Offline);

                    }
                    if (goingOffline==IDCANCEL)
                        CleanUp(rTicket,0);
                }
                lpInfo->fContinue = FALSE;
            }
            return 0;  // so goingOffline can be cleared with any other message
        }
        case OMSG_RENDEZVOUS_EVT_ONLINE:
        {
            LPRENDEZVOUSTICKET rTicket = (LPRENDEZVOUSTICKET)lpData;
            if (rTicket->hModule == lpOCMInfo->hModule) {
                lpInfo->fContinue = FALSE;
            }
            break;
        }
        case OMSG_RENDEZVOUS_EVT_ACCEPTED:
        {
            LPRENDEZVOUSTICKET rTicket = (LPRENDEZVOUSTICKET)lpData;
            if (rTicket->hModule == lpOCMInfo->hModule) {
                lpInfo->fContinue = FALSE;
            }
            break;
        }
    }
    goingOffline = 0;
    return 0;
}

//---------------------------------------------------------------------------
// Boilerplate OCM stuff
//---------------------------------------------------------------------------

BOOL API __export OCMOpen(LPOCMINFO lpOCM)
{
    lpOCMInfo = lpOCM;
    if(!OMRegister(OMTYPE_EVENT, OMGROUP_RENDEZVOUS, EventHandler))
```

```
        GROUPBOX           "When others issue the File Get command:" IDC_GROUP2,
                           6, 85, 185, 102
        CONTROL            "Allow no users to get my files",
                           IDC_ALLOW_NOONE,"Button",
                           BS_AUTORADIOBUTTON | WS_TABSTOP | WS_GROUP,14,93,170,14
        CONTROL            "Allow only users on my Buddy List to get my files",
                           IDC_ALLOW_BUDDY,"Button",
                           BS_AUTORADIOBUTTON | WS_TABSTOP,14,104,170,14
        CONTROL            "Allow everyone to get my files",IDC_ALLOW_ALL,"Button",
                           BS_AUTORADIOBUTTON | WS_TABSTOP,14,115,170,14
        LTEXT              "Directory from where others can get my files:",
                                          IDC_STATIC,14,130,160,12
        CONTROL            "",IDC_DIR_FILELIB,"Edit",
                           ES_AUTOHSCROLL | WS_BORDER | WS_GROUP | WS_TABSTOP,
                           14, 141, 171, 13
//                           14, 141, 115, 13
//        PUSHBUTTON         "B&rowse...",IDC_BROWSE_UPLOAD,
//                                                135,140,50,14,WS_GROUP | WS_TABSTOP
        CONTROL            "&Never display Status dialog",IDC_PUT_NO_STATUS,
                                          "Button", BS_AUTOCHECKBOX |
WS_TABSTOP,14,155,170,14
        CONTROL            "&Keep a record in logfile.txt of who has gotten files",
                                          IDC_PUT_LOG_FILES,
                                          "Button", BS_AUTOCHECKBOX |
WS_TABSTOP,14,169,170,14

```

```
FILELISTDLG DIALOG DISCARDABLE  10, 40, 270, 135
STYLE WS_POPUP | WS_DLGFRAME | WS_VISIBLE | WS_CAPTION | WS_SYSMENU
CAPTION "File Listing"
FONT 8, "MS Sans Serif"
BEGIN
    CONTROL         "",IDC_THERMO,"Static",SS_SIMPLE | WS_GROUP,8,3,192,39
    CTEXT           "",IDC_STATUS_TEXT1,8,12,234,27
    CONTROL         "Fast", IDC_IMFT_SPEED_FAST,"Button",
                    BS_AUTORADIOBUTTON | WS_TABSTOP | WS_GROUP,208,1,40,12
    CONTROL         "Medium", IDC_IMFT_SPEED_MEDIUM,"Button",
                    BS_AUTORADIOBUTTON | WS_TABSTOP,208,11,40,12
    CONTROL         "Slow", IDC_IMFT_SPEED_SLOW,"Button",
                    BS_AUTORADIOBUTTON | WS_TABSTOP,208,21,40,12
    CONTROL         "Pause", IDC_IMFT_SPEED_PAUSE,"Button",
                    BS_AUTORADIOBUTTON | WS_TABSTOP,208,31,40,11
  CONTROL         "",IDC_DIR_FILELIST, LISTBOX,
                                  LBS_EXTENDEDSEL|LBS_MULTIPLESEL|
                                  LBS_STANDARD|WS_VSCROLL|SBS_HORZ|WS_TABSTOP,
                                   8,45,254,68
    LTEXT           IDI_SENDFILE_GET, IDC_STATUS_ICON2, 10,114,32,32, SS_ICON
    DEFPUSHBUTTON   "Get",IDC_GET,42,117,51,14,WS_GROUP
    PUSHBUTTON      "Stop",IDABORT,107,117,51,14,WS_GROUP
    PUSHBUTTON      "Cancel",IDCANCEL,172,117,51,14,WS_GROUP
END
```

```
  FILETYPE        VFT_DLL
  FILESUBTYPE     VFT2_UNKNOWN
  FILEVERSION     0, 0, 0, 0

BEGIN
    BLOCK "StringFileInfo"
    BEGIN
        BLOCK VERSION_BUILD_TRANSLATION_STRING
        BEGIN
            VALUE "CompanyName",         VERSION_COMPANY
            VALUE "LegalCopyright",      VERSION_COPYRIGHT
            VALUE "ProductName",         VERSION_PRODUCT_NAME
            VALUE "ProductVersion",      VERSION_PRODUCT_VERSION_STRING
            VALUE "Build Number",        VERSION_BUILD_NUMBER

            // Module-specific info
            VALUE "FileDescription",     "Icbm File Transfer Module"
            VALUE "FileVersion",         "0.0.0.0"
            VALUE "InternalName",        "ICBMFT"
            VALUE "OriginalFilename",    "ICBMFT.OCM"
        END
    END

    BLOCK "VarFileInfo"
    BEGIN
        VALUE "Translation",  VERSION_BUILD_TRANSLATION
    END
END




++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

//
// (C) Copyright 1997  America Online, Inc. 75 Second Ave.
//                     Needham, MA 02194
//

#include "icbmft.h"
#include "string.h"

/* The sequence of events is:
   1.  Requester starts a Listen and sends REQUEST to buddy, with timeout
   1a. Requester times out --> Cancels
   2.  Receiver clicks "Accept" button, starts a Connect
   2a. Receiver Connect timesout; starts a Listen and sends ACCEPT to buddy
   2b. Sender gets ACCEPT, stops Listen and starts a Connect
   2c. Sender or Receiver timesout --> Cancels
   3.  Connection completed; start socket protocol
*/

void SockQuit(LPRENDEZVOUSTICKET rTicket)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (IS_RENDEZVOUS_DONE(rTicket)) {
    } else
        AppendMsg(rTicket,IDSFTP_RecvrCannotConnect,TRUE,FALSE,FALSE);
    SockCleanup(rTicket);
    CleanUp(rTicket, 0);
}

void SockStartQuitTimer(LPRENDEZVOUSTICKET rTicket)
```

```
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (!lpSrvStruct || IS_RENDEZVOUS_DONE(rTicket)) {
        SockQuit(rTicket);
        return;
    }

    // wait five seconds to give us enough time to get a CANCEL or NAK snac
    lpSrvStruct->sock_flags |= SockQuiting;
    lpSrvStruct->sock_timeout = GetTickCount() + 5000;
    o_SetTimer(rTicket->hDlgWnd, 102, 5000, NULL);
}

void SockStartWaitTimer(LPRENDEZVOUSTICKET rTicket)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (lpSrvStruct) {
        // keep connection open 10 minutes
        lpSrvStruct->sock_timeout = GetTickCount() + (DWORD)(60000*10);
        o_SetTimer(rTicket->hDlgWnd, 102, 60000, NULL);
    }
}

void SockListen(LPRENDEZVOUSTICKET rTicket)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (!lpSrvStruct)
        return;

    SockCleanup(rTicket);
    lpSrvStruct->sock_bufsize = SOCK_BUFSZ;

    lpSrvStruct->socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
    lpSrvStruct->sin_in.sin_family      = AF_INET;
    lpSrvStruct->sin_in.sin_port        = htons(rTicket->port);
    lpSrvStruct->sin_in.sin_addr.s_addr = INADDR_ANY;
    bind(lpSrvStruct->socket, (struct sockaddr *)&lpSrvStruct->sin_in,
         sizeof(lpSrvStruct->sin_in));
    WSAAsyncSelect(lpSrvStruct->socket, rTicket->hDlgWnd, WM_SOCKET,
                   FD_ACCEPT|FD_READ|FD_WRITE|FD_CLOSE);

    listen(lpSrvStruct->socket,1);
    lpSrvStruct->sock_flags = SockListening;
    lpSrvStruct->sock_timeout = 0;  // caller should set this if timeout
needed

    // we don't need a timer for requester because it's handled by IM window
}

void SockCleanup(LPRENDEZVOUSTICKET rTicket)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (lpSrvStruct) {
        if (lpSrvStruct->sock_flags &
SockListening|SockConnecting|SockConnected) {
            WSAAsyncSelect(lpSrvStruct->socket, rTicket->hDlgWnd, 0, 0);
            closesocket(lpSrvStruct->socket);
            lpSrvStruct->socket = 0;
        }
        lpSrvStruct->sock_flags = 0;
    }
}

```

```
void SockAcceptReady(LPRENDEZVOUSTICKET rTicket)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (!lpSrvStruct)
        return;

    if (lpSrvStruct->sock_flags & SockListening) {
        fd_set socks;
        struct timeval imtimeout;
        FD_ZERO(&socks);
        FD_SET(lpSrvStruct->socket,&socks);
        imtimeout.tv_sec=0; imtimeout.tv_usec=0;
        if (select(0,&socks,0,0,&imtimeout)) {
            int mm = sizeof(lpSrvStruct->sin_in);
            SOCKET soc;
            soc = accept(lpSrvStruct->socket,
                         (struct sockaddr *)&lpSrvStruct->sin_in, &mm);
            if (soc == INVALID_SOCKET) {
                SockQuit(rTicket);
                return;
            }
            closesocket(lpSrvStruct->socket);
            lpSrvStruct->socket = soc;
            lpSrvStruct->sock_flags &= ~SockListening;
            WSAAsyncSelect(lpSrvStruct->socket, rTicket->hDlgWnd, WM_SOCKET,
                           FD_READ|FD_WRITE|FD_CLOSE);
            lpSrvStruct->sock_flags |= SockConnected|SockReadyToReceiveHdr;
            rTicket->timeoutTime = 0;

            if (lpSrvStruct->type == TypeSend || lpSrvStruct->type == TypePut)
                SockSend(rTicket,TRUE);
            else
                SockRecvReady(rTicket);
        }
    }
}

void SockRecvReady(LPRENDEZVOUSTICKET rTicket)
{
    int n, num;
    DWORD nRecvd,chksum;
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    FTHDR *shdr = &lpSrvStruct->sock_hdr;
    if (!lpSrvStruct)
        return;

    lpSrvStruct->sock_flags |= SockReadyToReceive;
    if (lpSrvStruct->sock_flags & SockIsReceiving ||
        !(lpSrvStruct->sock_flags & SockConnected))
        return;
    lpSrvStruct->sock_flags |= SockIsReceiving;
    while (lpSrvStruct->sock_flags & SockReadyToReceive) {
        lpSrvStruct->sock_flags &= ~SockReadyToReceive;
        if (lpSrvStruct->sock_flags & SockReadyToReceiveHdr) {
            char buf[1024];
            LPSTR inptr;
            lpSrvStruct->speed_iter = 0;
            lpSrvStruct->speed_timestart = 0;
            if (!(lpSrvStruct->sock_flags & SockRecvingHdr)) {
                inptr = (LPSTR)shdr;
                num = sizeof(FTHDR);
            } else {
                // should never get here, unless other client has a filename
                // >256; throw these extra bytes away since filenames are
                // limited to 256 chars anyway
                inptr = buf;
                num = SWAP2(shdr->wHdrLen) - sizeof(FTHDR);
                if (num > sizeof(buf))
                    goto QUIT;
            }

//zzz TODO if FLAGS_ABORT, ignore non-header records
            n = recv(lpSrvStruct->socket, inptr, num, 0);
            if (n == SOCKET_ERROR) {
                int wsaerr = WSAGetLastError();
                if (wsaerr == WSAEWOULDBLOCK)
                    break;
                goto QUIT;
            } else if ((!(lpSrvStruct->sock_flags & SockRecvingHdr) &&
                        n < MIN_HDR_SZ) ||
                       ((lpSrvStruct->sock_flags & SockRecvingHdr) &&
                        n < num)) {
                goto QUIT;
            } else if (!(lpSrvStruct->sock_flags & SockRecvingHdr)) {
                if (FTValidateHdr(rTicket))
                    goto QUIT;
                if (n < SWAP2(shdr->wHdrLen)) {
                    lpSrvStruct->sock_flags |= 2430-2431
2444-2445SockRecvingHdr;
                    continue;
                }
            }
            lpSrvStruct->sock_flags &=
~(SockRecvingHdr|SockReadyToReceiveHdr);
            shdr->bName[FNSZ-1]=0;
            if (FTProcessHdr(rTicket))
                goto QUIT;

        } else if (lpSrvStruct->sock_flags & SockConnected) {
            // We are reading the file here
            long todo = SWAP4(shdr->dwFilesize) - SWAP4(shdr->dwNumRecvd);
            if (!todo)
                break;
            if (lpSrvStruct->status_numTodo == 0) {
                lpSrvStruct->status_numDone = 0;
                lpSrvStruct->status_numTodo = todo;
                lpSrvStruct->sock_starttime = GetTickCount();
                PaintThermo(rTicket,FALSE);
            }
            if (lpSrvStruct->speed == SpeedPause)
                break;
            else if (lpSrvStruct->speed_timewait) {
                DWORD delta = GetTickCount()-lpSrvStruct->speed_timestart;
                if (delta < lpSrvStruct->speed_timewait) {
                    delta = (lpSrvStruct->speed_timewait - delta);
                    if (delta > 0x7fff)
                        delta = 0x7fff;
                    lpSrvStruct->sock_flags |= SockRecvDelay;
                    o_SetTimer(rTicket->hDlgWnd, 102, delta, NULL);
                    break;
                }
            }

            num = (todo > SOCK_BUFSZ) ? SOCK_BUFSZ : (int)todo;
            n = recv(lpSrvStruct->socket, lpSrvStruct->sock_buf, num, 0);
            if (n == SOCKET_ERROR) {
                int wsaerr = WSAGetLastError();
                if (wsaerr == WSAEWOULDBLOCK)
                    break;
                SockStartQuitTimer(rTicket);
                return;
            } else if (n <= 0) {
                goto QUIT;
            }
            // write out what we just read
            if (fwrite(lpSrvStruct->sock_buf,n,1,lpSrvStruct->fileP) != 1) {
                // TODO: put up error-writing-file message
                goto QUIT;
            }

            nRecvd = SWAP4(shdr->dwNumRecvd) + n;
            shdr->dwNumRecvd = SWAP4(nRecvd);
            lpSrvStruct->sock_numSent = nRecvd;
            lpSrvStruct->sock_numTotal += n;
            lpSrvStruct->status_numDone += n;
            chksum = (shdr->dwRecvdChecksum);
            chksum = FTRecalcChecksum(chksum,(LPWORD)lpSrvStruct->sock_buf,n);
            shdr->dwRecvdChecksum = (chksum);
//zzz TODO if FLAGS_ABORT, pretend it's at the end
            if (lpSrvStruct->speed_iter == 0 || lpSrvStruct->speed_timewait) {
                lpSrvStruct->speed_timestart = GetTickCount();
            } else if (lpSrvStruct->speed_iter == SPEED_NUM_ITERS  &&
                       lpSrvStruct->speed_timewait == 0) {
                DWORD delta = (GetTickCount() -
                               lpSrvStruct->speed_timestart) /
SPEED_NUM_ITERS;
                lpSrvStruct->speed_timefor1 = delta;
                if (lpSrvStruct->speed == SpeedMedium)
                    lpSrvStruct->speed_timewait = delta * SPEED_MEDIUM;
                else if (lpSrvStruct->speed == SpeedSlow)
                    lpSrvStruct->speed_timewait = delta * SPEED_SLOW;
            }
            lpSrvStruct->speed_iter++;

            PaintThermo(rTicket,FALSE);
            if (SWAP4(shdr->dwNumRecvd) == SWAP4(shdr->dwFilesize)) {
                BOOL listData = FALSE;
                // finished with entire file; get ready for next one
                if (shdr->dwChecksum != 0 &&
                    shdr->dwChecksum != shdr->dwRecvdChecksum)
                    AppendMsg(rTicket,IDSFTP_RecvrDoneBadSum,FALSE,
                              ISFT_RCVR(lpSrvStruct),TRUE);
                else
                    AppendMsg(rTicket,IDSFTP_RecvrDone,FALSE,
                              ISFT_RCVR(lpSrvStruct),TRUE);
                FTCloseFileAndSetTime(rTicket);
                listData = (lpSrvStruct->state == StateListData);
                FTIncrementState(rTicket);
                SockSend(rTicket,TRUE);
                lpSrvStruct->sock_flags |= SockReadyToReceiveHdr;
                lpSrvStruct->doneNum++;
                lpSrvStruct->totalSizeOfDoneFiles += SWAP4(shdr->dwFilesize);
                if (lpSrvStruct->state == StateListWantToGet) {
                    FTInitFileList(rTicket, listData);
                    SET_RENDEZVOUS_DONE(rTicket);
                } else if (lpSrvStruct->doneNum >= lpSrvStruct->totalNum &&
                    lpSrvStruct->state != StateListWantToGet) {
                    SET_RENDEZVOUS_DONE(rTicket);
                    goto QUIT;
                    }
                }
            }
        }
        lpSrvStruct->sock_flags &= ~SockIsReceiving;
        return;
QUIT:
        SockQuit(rTicket);
        return;
}

void SockConnectComplete(LPRENDEZVOUSTICKET rTicket)
{
        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
        if (!lpSrvStruct)
            return;

        // MessageBox(0,"SockConnectComplete.",0,MB_OK);  // zzz
        lpSrvStruct->sock_timeout = 0;
        lpSrvStruct->sock_flags |= SockConnected|SockReadyToReceiveHdr;
        rTicket->timeoutTime = 0;
        if (lpSrvStruct->type == TypeSend || lpSrvStruct->type == TypePut)
            SockSend(rTicket,TRUE);
        else
            SockRecvReady(rTicket);
}

// return FALSE if we cannot connect
BOOL SockConnect(LPRENDEZVOUSTICKET rTicket)
{
        LPBYTE ipaddr = rTicket->ipAddrRemoteVerified;
        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
        if (!lpSrvStruct)
            goto err;

        if (ipaddr[0] == 0 || lpSrvStruct->sock_flags & SockTriedUnverified) {
            ipaddr = rTicket->ipAddrRemote;
            lpSrvStruct->sock_flags |= SockTriedUnverified;
        }
        if (ipaddr[0] == 0)
            goto err;

        lpSrvStruct->sock_bufsize = SOCK_BUFSZ;
        lpSrvStruct->sock_timeout = 0;

        if (!(lpSrvStruct->sock_flags & SockConnected)) {
            struct sockaddr_in addr;
            int ret;

            rTicket->timeoutTime = GetTickCount() + 60000;  // 1 minute

            addr.sin_family = AF_INET;
            addr.sin_port   = htons(rTicket->port);
            _fmemcpy(&addr.sin_addr, ipaddr, 4);
            if (lpSrvStruct->socket) {
                WSAAsyncSelect(lpSrvStruct->socket, rTicket->hDlgWnd, 0, 0);
                closesocket(lpSrvStruct->socket);
            }
            lpSrvStruct->socket = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
            WSAAsyncSelect(lpSrvStruct->socket, rTicket->hDlgWnd, WM_SOCKET,
                           FD_CONNECT|FD_READ|FD_WRITE|FD_CLOSE);
            ret = connect(lpSrvStruct->socket, (LPSOCKADDR)&addr, sizeof(addr));
            if (ret == SOCKET_ERROR) {
```

```
            int wsaerr = WSAGetLastError();
            if (wsaerr == WSAEWOULDBLOCK) {
                // start timer
                lpSrvStruct->sock_flags |= SockConnecting;
                lpSrvStruct->sock_timeout = GetTickCount() + 20000;
                o_SetTimer(rTicket->hDlgWnd, 102, 10000, NULL);
                return TRUE;
            }
            goto err;
        }
        lpSrvStruct->sock_flags |= SockConnected|SockReadyToReceiveHdr;
        rTicket->timeoutTime = 0;
        lpSrvStruct->sock_timeout = 0;
    }
    return TRUE;
err:
    SockQuit(rTicket);
    return FALSE;
}

void SockTimeout(LPRENDEZVOUSTICKET rTicket)
{
    DWORD ticks = GetTickCount();
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (!lpSrvStruct)
        return;

    if (lpSrvStruct->sock_timeout && (lpSrvStruct->sock_timeout < ticks))
    {
        if (lpSrvStruct->sock_flags & (SockClosing|SockQuiting)) {
        } else if (IS_RENDEZVOUS_TARGET(rTicket)) {
            if (lpSrvStruct->sock_flags & SockConnecting) {
                // cannot connect; start listen and ask buddy to connect to us
                SockListen(rTicket);
                SendCounter(rTicket);
                lpSrvStruct->sock_timeout = GetTickCount() + 20000;
                o_SetTimer(rTicket->hDlgWnd, 102, 10000, NULL);
                return;
            }
        }
        SockQuit(rTicket);
    } else if (lpSrvStruct->sock_timeout) {
        o_SetTimer(rTicket->hDlgWnd, 102, 10000, NULL);
    } else if (lpSrvStruct->sock_flags & SockSendDelay) {
        lpSrvStruct->sock_flags &= ~SockSendDelay;
        SockSendReady(rTicket);
    } else if (lpSrvStruct->sock_flags & SockRecvDelay) {
        lpSrvStruct->sock_flags &= ~SockRecvDelay;
        SockRecvReady(rTicket);
    }
}

void SockSend(LPRENDEZVOUSTICKET rTicket, BOOL sendhdr)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (!lpSrvStruct)
        return;

    if (!(lpSrvStruct->sock_flags & SockConnected))
        return;  // not connected, so we cannot send anything

    lpSrvStruct->sock_flags |= SockSending;
    if (sendhdr) {
        lpSrvStruct->sock_flags |= SockSendingHdr;
        FTInitHdr(rTicket);
    }
    SockSendReady(rTicket);
}

void SockSendReady(LPRENDEZVOUSTICKET rTicket)
{
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    FTHDR *shdr = &lpSrvStruct->sock_hdr;
    if (!lpSrvStruct)
        return;

    if (!(lpSrvStruct->sock_flags & (SockSendingHdr|SockSending))) {
        return;
    }
    while (1) {
        int n, num;
        if (lpSrvStruct->sock_flags & SockSendingHdr) {
            lpSrvStruct->speed_iter = 0;
            lpSrvStruct->speed_timestart = 0;
            num = SWAP2(shdr->wHdrLen);
            n = send(lpSrvStruct->socket, (LPSTR)shdr,num,0);
            if (n == SOCKET_ERROR) {
                int wsaerr = WSAGetLastError();
                if (wsaerr == WSAEWOULDBLOCK)
                    return;
                goto QUIT;
            } else if (n < num) {
                goto QUIT;
            }
            break;
        } else if (lpSrvStruct->sock_flags & SockSending) {
            long lnum = SWAP4(shdr->dwFilesize) -
                lpSrvStruct->sock_numSent;
            if (lpSrvStruct->status_numTodo == 0) {
                lpSrvStruct->status_numDone = 0;
                lpSrvStruct->status_numTodo = lnum;
                lpSrvStruct->sock_starttime = GetTickCount();
            }
again:
            num = (lnum > (long)lpSrvStruct->sock_bufsize) ?
                lpSrvStruct->sock_bufsize : (int)lnum;

            if (num) {
                fseek(lpSrvStruct->fileP, lpSrvStruct->sock_numSent,
SEEK_SET);

                if (fread(lpSrvStruct->sock_buf,num,1,lpSrvStruct->fileP) < 1)
                    goto QUIT;

                if (lpSrvStruct->speed == SpeedPause)
                    return;
                else if (lpSrvStruct->speed_timewait) {
                    DWORD delta = GetTickCount()-lpSrvStruct->speed_timestart;
                    if (delta < lpSrvStruct->speed_timewait) {
                        delta = (lpSrvStruct->speed_timewait - delta);
                        if (delta > 0x7fff)
                            delta = 0x7fff;
                        lpSrvStruct->sock_flags |= SockSendDelay;
                        o_SetTimer(rTicket->hDlgWnd, 102, delta, NULL);
                        return;
                    }
                }
                n = send(lpSrvStruct->socket, lpSrvStruct->sock_buf, num, 0);
                if (n == SOCKET_ERROR) {
                    int wsaerr = WSAGetLastError();
                    if (wsaerr == WSAEWOULDBLOCK)
                        return;
                    else if (wsaerr == WSAEMSGSIZE &&
                             lpSrvStruct->sock_bufsize > 16) {
                        // 16 above prevents infinite loops
                        lpSrvStruct->sock_bufsize >>= 1;
                        goto again;
                    }
                    goto QUIT;
                } else {
                    lpSrvStruct->sock_numSent += n;
                    lpSrvStruct->sock_numTotal += n;
                    lpSrvStruct->status_numDone += n;
                    if (lpSrvStruct->speed_iter == 0 ||
                        lpSrvStruct->speed_timewait) {
                        lpSrvStruct->speed_timestart = GetTickCount();
                    } else if (lpSrvStruct->speed_iter == SPEED_NUM_ITERS &&
                              lpSrvStruct->speed_timewait == 0) {
                        DWORD delta = ((GetTickCount() -
                                      lpSrvStruct->speed_timestart) /
                                     SPEED_NUM_ITERS);
                        lpSrvStruct->speed_timefor1 = delta;
                        if (lpSrvStruct->speed == SpeedMedium)
                            lpSrvStruct->speed_timewait = delta *
SPEED_MEDIUM;
                        else if (lpSrvStruct->speed == SpeedSlow)
                            lpSrvStruct->speed_timewait = delta * SPEED_SLOW;
                    }
                    lpSrvStruct->speed_iter++;
                }
                PaintThermo(rTicket,FALSE);
            }
            if (lpSrvStruct->sock_numSent == SWAP4(shdr->dwFilesize))
                break;
        } else
            return;
    }

    lpSrvStruct->sock_flags &= ~(SockSendingHdr|SockSending);
    FTIncrementState(rTicket);
    if (lpSrvStruct->state != StateFileData &&
        lpSrvStruct->state != StateListData)
        lpSrvStruct->sock_flags |= SockReadyToReceiveHdr;
    return;  // next wait for a reply

QUIT:
    SockStartQuitTimer(rTicket);
    return;

}

void SockConnectionClosed(LPRENDEZVOUSTICKET rTicket)
{
    // delay before calling SockQuit() to give time to receive a NAK or CANCEL
    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (lpSrvStruct) {
        lpSrvStruct->sock_flags |= SockClosing;
        SockStartQuitTimer(rTicket);
    }
```

```
}

LRESULT SockMessage(LPRENDEZVOUSTICKET rTicket, WPARAM wParam, LPARAM lParam)
{
    LRESULT result = 1;

    WORD event = WSAGETSELECTEVENT(lParam);
    WORD error = WSAGETSELECTERROR(lParam);

    LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
    if (lpSrvStruct && (lpSrvStruct->sock_flags & SockClosing)) {
        // ignore all socket events because we are waiting for timeout
    }
    else if (error == 0)
    {
        if (event & FD_ACCEPT)
            SockAcceptReady(rTicket);
        if (event & FD_CONNECT)
            SockConnectComplete(rTicket);
        if (event & FD_READ)
            SockRecvReady(rTicket);
        if (event & FD_WRITE)
            SockSendReady(rTicket);
        if (event & FD_CLOSE)
            SockConnectionClosed(rTicket);
    }
    else
    {
        LPSRVSTRUCT lpSrvStruct = (LPSRVSTRUCT)rTicket->lpSrvStruct;
        switch (error)
        {
          case WSAECONNREFUSED:
          case WSAECONNABORTED:
          case WSAECONNRESET:
          case WSAENETDOWN:
          default:
            {
              int flags = lpSrvStruct->sock_flags;
              if (!(flags & (SockTriedUnverified|SockListening)) &&
                  (flags & SockConnecting)) {
                  lpSrvStruct->sock_flags |= SockTriedUnverified;
                  SockConnect(rTicket);
              } else if (!(lpSrvStruct->sock_flags & SockConnected)) {
                  lpSrvStruct->sock_timeout = GetTickCount() - 1;
                  SockTimeout(rTicket);
              } else {
                  SockStartQuitTimer(rTicket);
              }
              result = 0;
            }
        }
    }
    return result;
}
```

```
void connLookupHost(LPCONNECTION c, LPCSTR host, LPHOSTENT hostent)
{
    /* If the host is specified with an IP address, stash the address into
     * the hostent buffer and fake a lookup-complete event.
     */
    if (host[0] >= '0' && host[0] <= '9')
    {
        DWORD iaddr;

        if ((iaddr = inet_addr(host)) == INADDR_NONE)
        {
            ConnDisconnect(c);
            connCallFatalError(c, c->session, IDS_BAD_ADDRESS, NULL);
            return;
        }

        connBuildFakeHostEnt(hostent, host, iaddr);
        connEventLookupComplete(c);
```

Exhibit 3

```
    }

    /* If host was specified with a hostname, initiate an asynchronous
     * DNS lookup.  This will culminate with either an EventLookupComplete()
     * or an EventLookupFailed() call.  Only do a timeout in the case of
     * a proxy lookup.  A timeout in the case of a host lookup does not
     * work with the "auto dial" feature of Windows 95; it does not allow
     * enough time to dial and make the connection.  Also, the error message
     * for DNS lookup is confusing to users.
     */
    else
    {
        if (c->state == CONN_STATE_PROXY_LOOKUP)
            o_SetTimer(c->hSockWnd, TIMER_ID_LOOKUP, TIMEOUT_LOOKUP, NULL);
        c->hLookupTask = WSAAsyncGetHostByName(c->hSockWnd,
                                               WM_GETHOSTBYNAME, host,
                                               (LPBYTE)hostent,
                                               MAXGETHOSTSTRUCT);
    }
}
```

```
void connConnectToHost(LPCONNECTION c, LPVOID ipaddr, WORD port)
{
    SOCKADDR_IN addr;

    connDestroySocket(c);
    connCreateSocket(c);

    addr.sin_family = AF_INET;
    addr.sin_port = htons(port);
    _fmemcpy(&addr.sin_addr, ipaddr, 4);
    connect(c->sock, (LPSOCKADDR)&addr, sizeof(addr));
    o_SetTimer(c->hSockWnd, TIMER_ID_CONNECT, TIMEOUT_CONNECT, NULL);

    TRACECONNECT(inet_ntoa(addr.sin_addr), port, c->sock);
}
```

```
void connDoServerLookup(LPCONNECTION c)
{
    connChangeState(c, CONN_STATE_LOOKUP);
    connLookupHost(c, c->serv->host, c->oscarHostEnt);
}
```

```
void connDoServerConnect(LPCONNECTION c)
{
    connChangeState(c, CONN_STATE_CONNECT);

    /* If there is no port, don't proceed.  The owner will
     * disconnect us.
     */
    if (c->serv->port == 0)
        return;

    connConnectToHost(c, &c->ipaddr, c->serv->port);
}
```

```
void connDoValidation(LPCONNECTION c)
{
    /* Only proceed if there is authorization.  If there is no
     * authorization, hang in this state until the owner explicitly
     * disconnects us.  This is used for auto-config.
     */
    if (c->auth != NULL)
    {
        /* Send the SIGNON packet.
         */
        connSendSignOn(c);

        /* Nuke the auth structure.  It's no longer needed once the
         * SIGNON packet is sent.
         */
        if (c->auth)
        {
            MemFree(c->auth);
            c->auth = NULL;
        }
    }

    connChangeState(c, CONN_STATE_VALIDATE);
}
```

```
void connEventLookupComplete(LPCONNECTION c)
{
    int naddrs, index;

    o_KillTimer(c->hSockWnd, TIMER_ID_LOOKUP);
    c->hLookupTask = 0;

    switch (c->state)
    {
      case CONN_STATE_LOOKUP:
      {
          /* More than one address may be returned from the DNS
           * lookup.  Count how many there are and choose one at
           * random (for load distribution).
           */
          naddrs = 0;
          while (c->oscarHostEnt->h_addr_list[naddrs])
              naddrs++;
          index = naddrs == 1 ? 0 : (int)((GetTickCount()/1000) % naddrs);

          /* Save these in case the attempt to connect to the first
           * host fails.  In this case, we will want to try other hosts.
           */
          c->numHosts = naddrs;
          c->initialHostIndex = c->currentHostIndex = index;
          _fmemcpy(&c->ipaddr,
                   c->oscarHostEnt->h_addr_list[c->currentHostIndex],
                   sizeof(IN_ADDR));

          if (c->prox != NULL && c->prox->useProxy)
```

```
                connDoProxyLookup(c);
            else
                connDoServerConnect(c);
            break;
        }

        case CONN_STATE_PROXY_LOOKUP:
        {
            connDoProxyConnect(c);
            break;
        }
    }
}
```

```
void connEventAwaitChallangeComplete(LPCONNECTION c)
{
    o_KillTimer(c->hSockWnd, TIMER_ID_CHALLANGE);
    connDoValidation(c);
}
```

```
        case WM_GETHOSTBYNAME:
        {
            WORD error = WSAGETASYNCERROR(lParam);
            if (error == 0)
                connEventLookupComplete(c);
            else
                connEventLookupFailed(c);
            break;
        }
```

```
                if (event & FD_READ)
                {
                    if (c->state <= CONN_STATE_PROXY_REQUEST)
                    {
                        switch (c->prox->protocol)
                        {
                          case DLG_PROX_PROTO_SOCKS4:
```

```
                case DLG_PROX_PROTO_SOCKS5:
                    connEventSocksResponse(c);
                    break;
                case DLG_PROX_PROTO_HTTPS:
                    connEventHttpsResponse(c);
                    break;
            }
        }
        else
            connEventRecvReady(c);
}
```

```
LPCONNECTION ConnCreate(LPVOID owner, CONNCALLBACK callback)
{
    LPCONNECTION c;

    if ((c = (LPCONNECTION)MemAlloc(sizeof(CONNECTION))) == NULL)
        return NULL;

    if (!(c->hSockWnd = CreateWindow(CONN_CLASS, NULL, WS_POPUP,
                                     0, 0, 0, 0,
                                     NULL, NULL, lpOCMInfo->hModule, NULL)))
    {
        MemFree(c);
        return NULL;
    }
    SetWindowLong(c->hSockWnd, 0, (LONG)c);

    c->callback         = callback;

    if ((c->session = (LPSESSION)owner) != NULL)
        connCallInsertConnection(c, c->session);

    c->state            = CONN_STATE_OFFLINE;
    c->sock             = INVALID_SOCKET;
    c->hLookupTask      = 0;
    c->oscarHostEnt     = NULL;
    c->proxyHostEnt     = NULL;
    c->serv             = NULL;
    c->prox             = NULL;
    c->auth             = NULL;
    c->proxResp         = NULL;
    c->destroyed        = FALSE;
    c->isReceiving      = FALSE;
    c->isBOSConnection  = FALSE;
    c->numHosts         = 0;
    c->initialHostIndex = 0;
    c->currentHostIndex = 0;
    c->numUnstartedSrvs = 0;
    c->numServices      = 0;
    c->serviceService   = NULL;
    c->meterMap          = MapCreate();
    c->nInactivePeriods = 0;
    c->shutdownCount     = 0;

    INIT_LIST(&c->services);
    INIT_LIST(&c->meters);

    connInitSend(c);
    connInitRecv(c);

    return c;
}
```

```
void ConnConnect(LPCONNECTION c, LPSERVCONFIG serv, LPPROXCONFIG prox,
                 LPAUTHINFO auth)
{
    /* Reinitialize sending and receiving FSMs.
     */
    connInitSend(c);
    connInitRecv(c);

    /* Save server info.
     */
    if (serv != NULL)
    {
        if (c->serv == NULL)
            c->serv = (LPSERVCONFIG)MemAlloc(sizeof(SERVCONFIG));
        _fmemcpy(c->serv, serv, sizeof(SERVCONFIG));
    }

    /* Save proxy info.
     */
    if (prox != NULL)
    {
        if (c->prox == NULL)
            c->prox = (LPPROXCONFIG)MemAlloc(sizeof(PROXCONFIG));
        _fmemcpy(c->prox, prox, sizeof(PROXCONFIG));
    }

    /* Save the authorization data until we are successfully connected.
     */
    if (auth != NULL)
    {
        if (c->auth == NULL)
            c->auth = (LPAUTHINFO)MemAlloc(sizeof(AUTHINFO));
        _fmemcpy(c->auth, auth, sizeof(AUTHINFO));

        /* Assume it's the BOS connection if we have a username/password
         * authorization.  We need to set this immediately so that sess.c
         * doesn't screen out fatal errors during signon.
         */
        if (auth->type == AUTH_TYPE_USER)
            c->isBOSConnection = TRUE;
    }
    else
    {
        if (c->auth != NULL)
        {
            MemFree(c->auth);
            c->auth = NULL;
        }
    }

    c->oscarHostEnt = (LPHOSTENT)MemAlloc(MAXGETHOSTSTRUCT);

    /* If we're not using a proxy to do hostname resolution, initiate
     * a DNS lookup of the server.  Otherwise start the proxy connect
     * sequence by doing a DNS lookup of the proxy.
     */
    if (c->prox == NULL || !c->prox->useProxy ||
        (!c->prox->resolveHostnames &&
         (c->prox->protocol == DLG_PROX_PROTO_SOCKS4 ||
          c->prox->protocol == DLG_PROX_PROTO_SOCKS5)))
        connDoServerLookup(c);
    else
        connDoProxyLookup(c);
}
```

```
void connReceiveBlock(LPCONNECTION c)
{
    LPRECVSTATE r = &c->recvState;
    int n;
    BYTE buf[512];
    LPBYTE ptr;

    if ((n = recv(c->sock, (char*)buf, 512, 0)) == SOCKET_ERROR)
        return;
    c->nInactivePeriods = 0;

    ptr = buf;

    while (n-- > 0)
    {
        BYTE b = *ptr++;
```

```
                 * buffer.
                 */
                bytesToCopy = min(r->bytesLeft, (WORD)n);
                _fmemcpy(r->currentByte, ptr, bytesToCopy);

                /* Adjust counts and pointers accordingly.
                 */
                n -= bytesToCopy;
                ptr += bytesToCopy;
                r->bytesLeft -= bytesToCopy;
                r->currentByte += bytesToCopy;

                /* If the packet is all here, dispatch it and begin
                 * waiting for a new packet.
                 */
                if (r->bytesLeft == 0)
                {
                    TRACERECVFLAP(inet_ntoa(c->ipaddr), r->type, r->seqNumber,
                                  r->length, r->data);
                    connProcessFLAP(c, r->type, r->length, r->data);
                    MemFree(r->data);
                    r->data = 0;
                    r->state = RECV_STATE_UNKNOWN;
                }
                break;
            }
        }
    }
}


void connEventRecvReady(LPCONNECTION c)
{
    LPRECVSTATE r = &c->recvState;

    /* Indicate that there's something to receive.
     */
    r->readyToReceive = TRUE;

    /* Block reentry to the actual receiving code.  Reentry can happen
     * if a SNAC handler puts up a modal dialog box.
     */
    if (c->isReceiving)
        return;
    c->isReceiving = TRUE;

    /* As long as data is available, read and process it.  Clear the
     * ready flag before processing, though, because it can be reset
     * during processing if a SNAC handler puts up a modal dialog.
     */
    while (r->readyToReceive)
    {
        r->readyToReceive = FALSE;
        connReceiveBlock(c);

        /* If processing the received block caused the connection
         * to be destroyed, finish the job now and return immediately
         * without referencing the connection object again.
         */
        if (c->destroyed)
        {
            MemFree(c);
            return;
        }
    }

    /* Allow entry again.
     */
    c->isReceiving = FALSE;
}
```

```
void connEventSendReady(LPCONNECTION c)
{
    LPSENDSTATE s = &c->sendState;
    LPITEM item;

    s->readyToSend = TRUE;
    while ((item = FIRST_ITEM(&s->queue)) != NULL_ITEM(&s->queue))
    {
        LPPACKET  packet = (LPPACKET)item;
        LPSERVICE service = PacketService(packet);
        LPQUEUE serviceQueue = PacketQueue(packet);
        WORD bytesAvailable;
        int bytesSent;  // must be signed to detect error return from send()
        LPBYTE addr;

        /* Attempt to send as much of the block as possible.  Also
         * indicate that there has been activity on the connection.
         */
        PacketGetData(packet, &bytesAvailable, &addr);
        bytesSent = send(c->sock, addr, bytesAvailable, 0);
        c->nInactivePeriods = 0;

        /* There is a chance that Winsock isn't really ready.  This
         * could happen if some other app snuck in and sent some data
         * before us.  It could also happen because of buggy Winsock
         * stacks (like the Shiva dialer from Netscape).  In any case,
         * the correct thing to do is to try again.
         */
        if (bytesSent == SOCKET_ERROR)
        {
            TraceMsg(TRACE_PROTO, "PROTO: Error sending data...retrying");
            return;
        }

        /* If only part of the packet could be sent, update its internal
         * pointers and return.  We will be called again when WinSock is
         * ready to accept more data.
         */
        if (bytesSent < (int)bytesAvailable)
        {
            PacketAdvance(packet, (WORD)bytesSent);
            s->readyToSend = FALSE;
            return;
        }

        /* If the entire packet has been sent, destroy it.  This will
         * also remove it from its queue.  It's important to do this before
         * notifying the service, because the notification could cause
         * another packet to be queued, which could, in turn, cause this
         * to be reentered and send the old packet again.
         */
        PacketDestroy(packet);

        /* Finally notivy the service.
         */
        if (service)
            ServPacketSent(service, serviceQueue);
    }
}


void ConnSendPacket(LPCONNECTION c, LPPACKET packet)
{
    LPSENDSTATE s = &c->sendState;

    PacketSetSequenceNumber(packet, s->seqNumber++);

    TRACESENDFLAP(inet_ntoa(c->ipaddr),
                  PacketType(packet), PacketSequenceNumber(packet),
                  PacketContentSize(packet), PacketContentAddr(packet));

    INSERT_ITEM_AT_TAIL((LPITEM)packet, &s->queue);

    /* If the socket can accept data, attempt to send the block now.
     */
    if (s->readyToSend)
        connEventSendReady(c);
}
```

```
void SessSignOn(LPSESSION s, LPCSTR nickname, LPCSTR password)
{
    LPCONNECTION connection;
    AUTHINFO authinfo;

    /* Ignore this request if we are already signed on or are in the
     * process of signing on.
     */
    if (s->state > OM_PROTO_STATE_OFFLINE)
      return;

    /* Create the initial connection.  It will initially be used
     * for authorization, but will ultimately become the first BOS
     * connection.
     */
    if ((connection = ConnCreate(s, sessConnCallback)) == NULL)
        return;

    /* Establish the connection, using the supplied nickname and password
     * for authorization.
     */
    authinfo.type = AUTH_TYPE_USER;
    o_strncpy(authinfo.user.nickname, nickname, MAX_SZ_NICKNAME_LEN);
    o_strncpy(authinfo.user.password, password, MAX_SZ_PASSWORD_LEN);

    ConnConnect(connection, &DlgServConfig, &DlgProxConfig, &authinfo);

    /* Save the nickname so that other OCMs can enquire about it.  This
     * is temporary.  It will be replaced by the official nickname (with
     * capitalization and spacing from registration database) once we
     * retrieve that from the server.
     */
    SessSetNickname(s, nickname);
}
```

```
void connSendSignOn(LPCONNECTION c)
{
    LPPACKET packet;
    BYTE verbuf[128], buf[512];
    WORD len;
    SNACSTREAM ss;
    LPAUTHINFO auth = c->auth;

    SNACOpen(&ss, sizeof buf, buf);

    SNACPutLong(&ss, 1);

    switch (auth->type)
    {
      case AUTH_TYPE_USER:
      {
          WORD i;
          long temp;
          WORD nicksize = lstrlen(auth->user.nickname);
          WORD passsize = lstrlen(auth->user.password);

          if (nicksize == 0 || passsize == 0)
          {
              connDisconnect(c);
              connCallFatalError(c, c->session, IDS_PASSWORD_REQUIRED, NULL);
              return;
          }

          /* Send nickname and password.
           */
          SNACPutWord(&ss, TLV_TAGS_NICK);
          SNACPutWord(&ss, nicksize);
          SNACPutBlock(&ss, nicksize, (LPBYTE)auth->user.nickname);

          SNACPutWord(&ss, TLV_TAGS_PASSWORD);
          SNACPutWord(&ss, passsize);
          for (i = 0; i < passsize; i++)
              SNACPutByte(&ss, (BYTE)(auth->user.password[i]^xorstring[i]));

          /* Send client version string.
           */
          len = (WORD)wsprintf(verbuf, "%s, version %s/%s",
                      (LPCSTR)VERSION_PRODUCT_NAME,
                      (LPCSTR)VERSION_PRODUCT_VERSION_STRING,
                      VERSION_FILEOS == VOS_DOS_WINDOWS16 ?
                                   (LPCSTR)"WIN16" : (LPCSTR)"WIN32");
          SNACPutWord(&ss, TLV_TAGS_CLIENT_IDENTITY);
          SNACPutWord(&ss, len);
          SNACPutBlock(&ss, len, verbuf);

          /* Send structured client version.
           */
          {
              WORD maj, min, pnt, bld;

              ParseVersionNumber(&maj, &min, &pnt, &bld);

              SNACPutWord(&ss, TLV_TAGS_CLIENT_ID);
              SNACPutWord(&ss, 2);
              SNACPutWord(&ss, TOOL_ID);

              SNACPutWord(&ss, TLV_TAGS_MAJOR_VERSION);
```

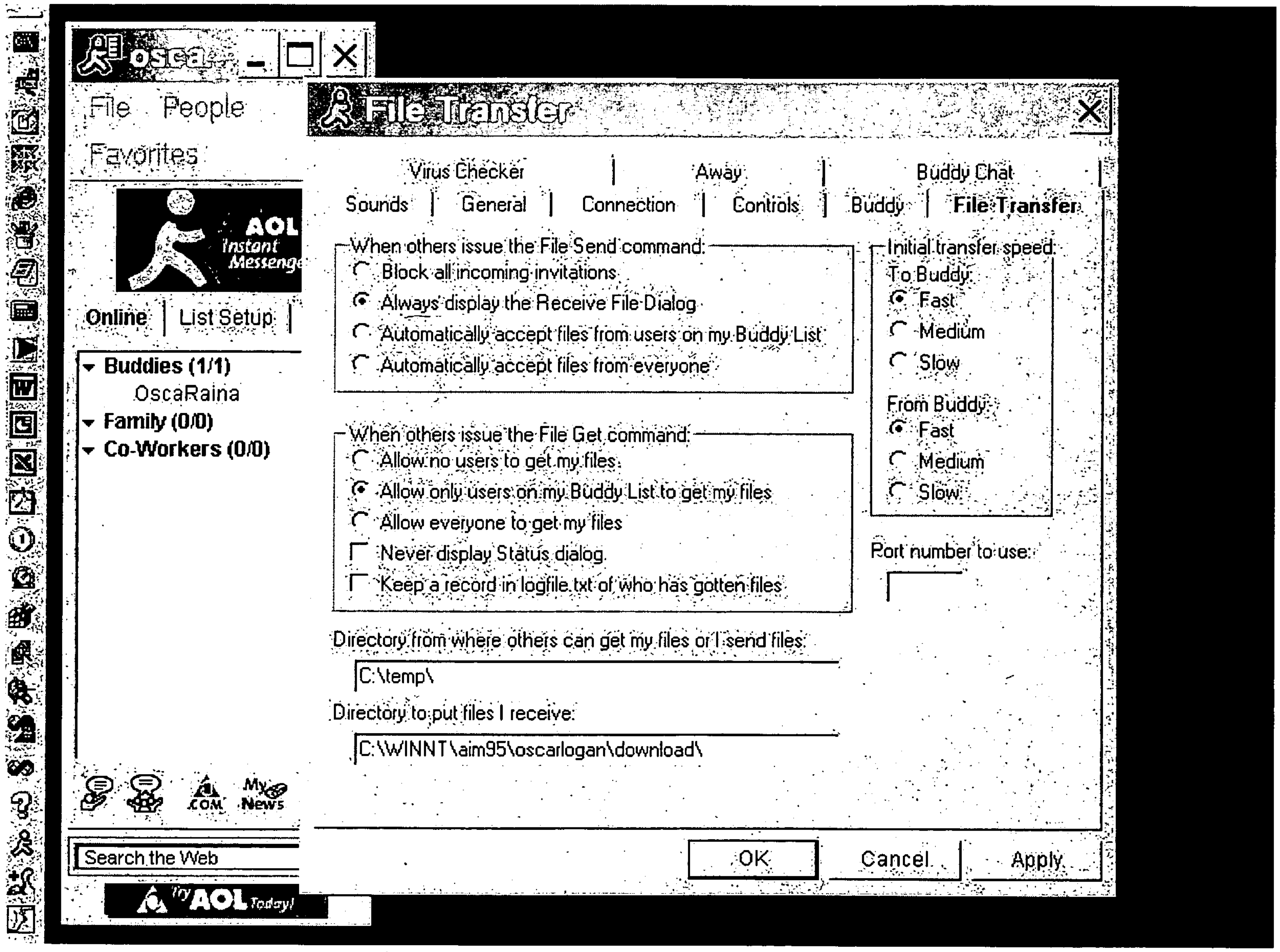
```
                SNACPutWord(&ss, 2);
                SNACPutWord(&ss, maj);

                SNACPutWord(&ss, TLV_TAGS_MINOR_VERSION);
                SNACPutWord(&ss, 2);
                SNACPutWord(&ss, min);

                SNACPutWord(&ss, TLV_TAGS_POINT_VERSION);
                SNACPutWord(&ss, 2);
                SNACPutWord(&ss, pnt);

                SNACPutWord(&ss, TLV_TAGS_BUILD_NUM);
                SNACPutWord(&ss, 2);
                SNACPutWord(&ss, bld);
            }

            /* Send the international crap.
             */
            {
                char buf[32];

                if (RDBLoadString(RESMODULE, IDS_INTL_COUNTRY_CODE,
                                  buf, sizeof buf))
                {
                    SNACPutWord(&ss, TLV_TAGS_COUNTRY);
                    SNACPutString(&ss, buf);
                }
                if (RDBLoadString(RESMODULE, IDS_INTL_LANGUAGE_CODE,
                                  buf, sizeof buf))
                {
                    SNACPutWord(&ss, TLV_TAGS_LANGUAGE);
                    SNACPutString(&ss, buf);
                }
                if (RDBLoadString(RESMODULE, IDS_INTL_SCRIPT_CODE,
                                  buf, sizeof buf))
                {
                    SNACPutWord(&ss, TLV_TAGS_SCRIPT);
                    SNACPutString(&ss, buf);
                }
            }

            /* If it's a non-AOL client, send the distribution channel
             */
            temp = RDBLoadValue(RESMODULE, IDV_DIST_CHANNEL, 0);
            if (temp != 0)
            {
                SNACPutWord(&ss, TLV_TAGS_DIST_CHANNEL);
                SNACPutWord(&ss, 4);
                SNACPutLong(&ss, (DWORD)temp);
            }

            /* If there is a stored disconnect reason, send it now.
             */
            temp = ProfGetLong(PROF_GLOBAL, MISC_GROUP, MISC_KEY_DISCONNECT);
            if (temp != 0)
            {
                SNACPutWord(&ss, TLV_TAGS_DISCONNECT_REASON);
                SNACPutWord(&ss, 2);
                SNACPutWord(&ss, (WORD)temp);
            }

            /* Mark this connection as the BOS connection.
             */
            c->isBOSConnection = TRUE;

            break;
        }
        case AUTH_TYPE_COOKIE:
        {
            SNACPutWord(&ss, TLV_TAGS_LOGIN_COOKIE);
            SNACPutWord(&ss, auth->cookie.length);
            SNACPutBlock(&ss, auth->cookie.length, auth->cookie.buffer);
            break;
        }
    }


    packet = PacketCreate(FLAP_SIGNON_TYPE, SNACBytesTransferred(&ss), buf,
                          NULL, NULL);
    ConnSendPacket(c, packet);
}
```

```
void connProcessSignOn(LPCONNECTION c, LPSNACSTREAM lpss)
{
    DWORD version;
    SNACGetLong(lpss, &version);
    if (version == 1)
        connEventAwaitChallangeComplete(c);
    else
        connEventAwaitChallangeFailed(c);
}
```

```
void connProcessFLAP(LPCONNECTION c, WORD type, WORD size, LPBYTE data)
{
    SNACSTREAM ss;

    SNACOpen(&ss, size, data);

    switch (type)
    {
      case FLAP_SIGNON_TYPE:
          connProcessSignOn(c, &ss);
          break;

      case FLAP_SIGNOFF_TYPE:
          connProcessSignoff(c, &ss);
          break;

      case FLAP_ERROR_TYPE:
          connProcessError(c, &ss);
          break;

      case FLAP_DATA_TYPE:
      {
          WORD  group, type, flags;
          DWORD reqid;
          SNACGetHeader(&ss, &group, &type, &flags, &reqid);
          if (group == OMGROUP_SERVICE)
          {
              switch (type)
              {
                case SNAC_SNAC_ERR:
                {
                    WORD code, type, len;

                    SNACGetWord(&ss, &code);
                    while (SNACGetWord(&ss, &type))
                    {
                        SNACGetWord(&ss, &len);
                        SNACSkipBytes(&ss, len);
                    }

                    switch (GETTAG(reqid))
                    {
                      case SNAC_SERVICE_CLIENT_ONLINE:
```

```
                    break;
                case SNAC_SERVICE_MOTD:
                    connProcessMOTD(c, &ss, reqid);
                    break;
            }
        }
        else
        {
            OMSendMessage(OMTYPE_SNAC, group, type, size, data);
        }
        break;
    }
  }
}
```

BEST AVAILABLE COPY



When others issue the File Send command:
- ( ) Block all incoming invitations
- (●) Always display the Receive File Dialog
- ( ) Automatically accept files from users on my Buddy List
- ( ) Automatically accept files from everyone

When others issue the File Get command:
- ( ) Allow no users to get my files
- (●) Allow only users on my Buddy List to get my files
- ( ) Allow everyone to get my files
- [ ] Never display Status dialog
- [ ] Keep a record in logfile.txt of who has gotten files

Initial transfer speed:
To Buddy:
- (●) Fast
- ( ) Medium
- ( ) Slow

From Buddy:
- (●) Fast
- ( ) Medium
- ( ) Slow

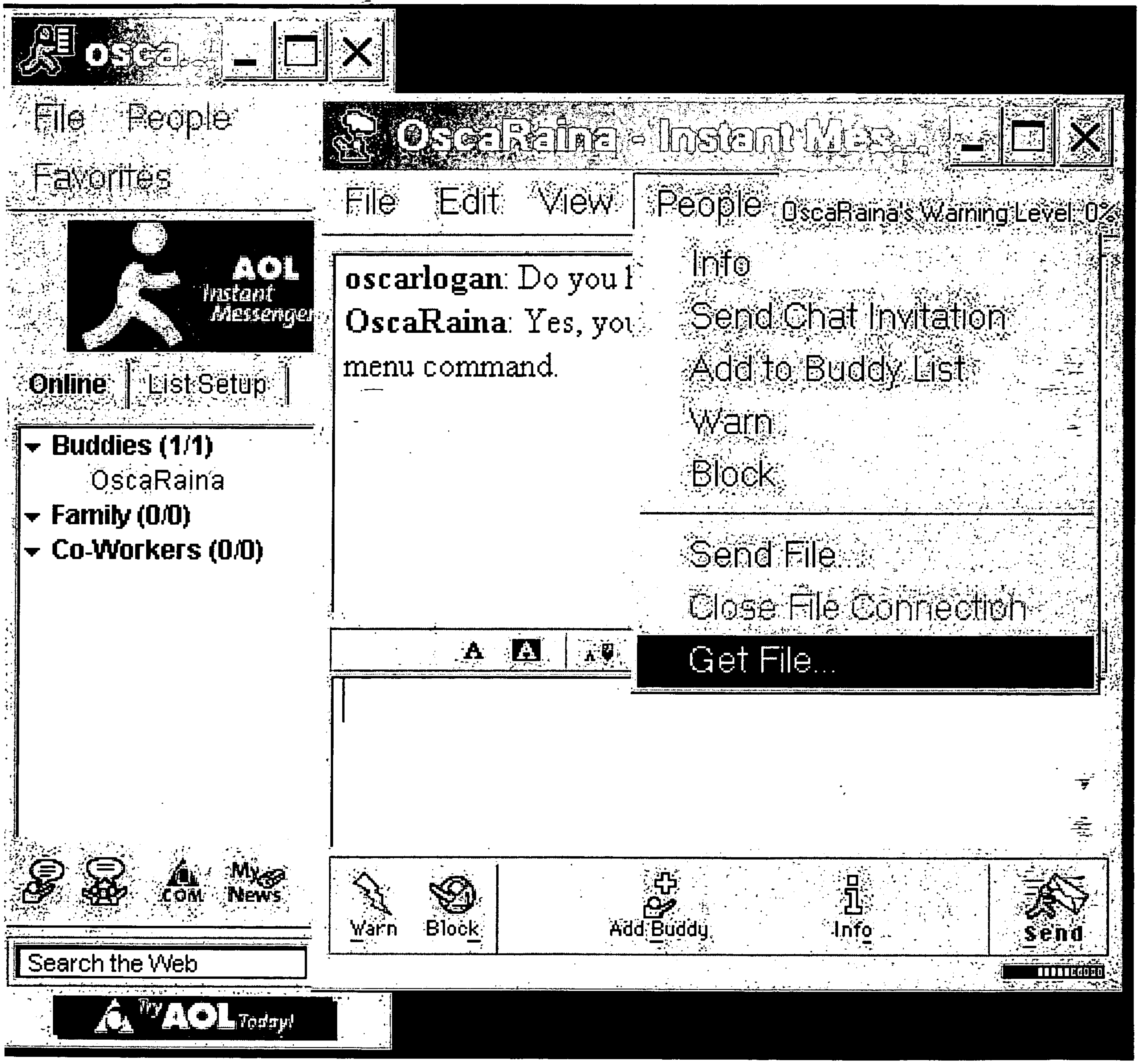
Port number to use:

Directory from where others can get my files or I send files:
C:\temp\

Directory to put files I receive:
C:\WINNT\aim95\oscarlogan\download\

Exhibit 4

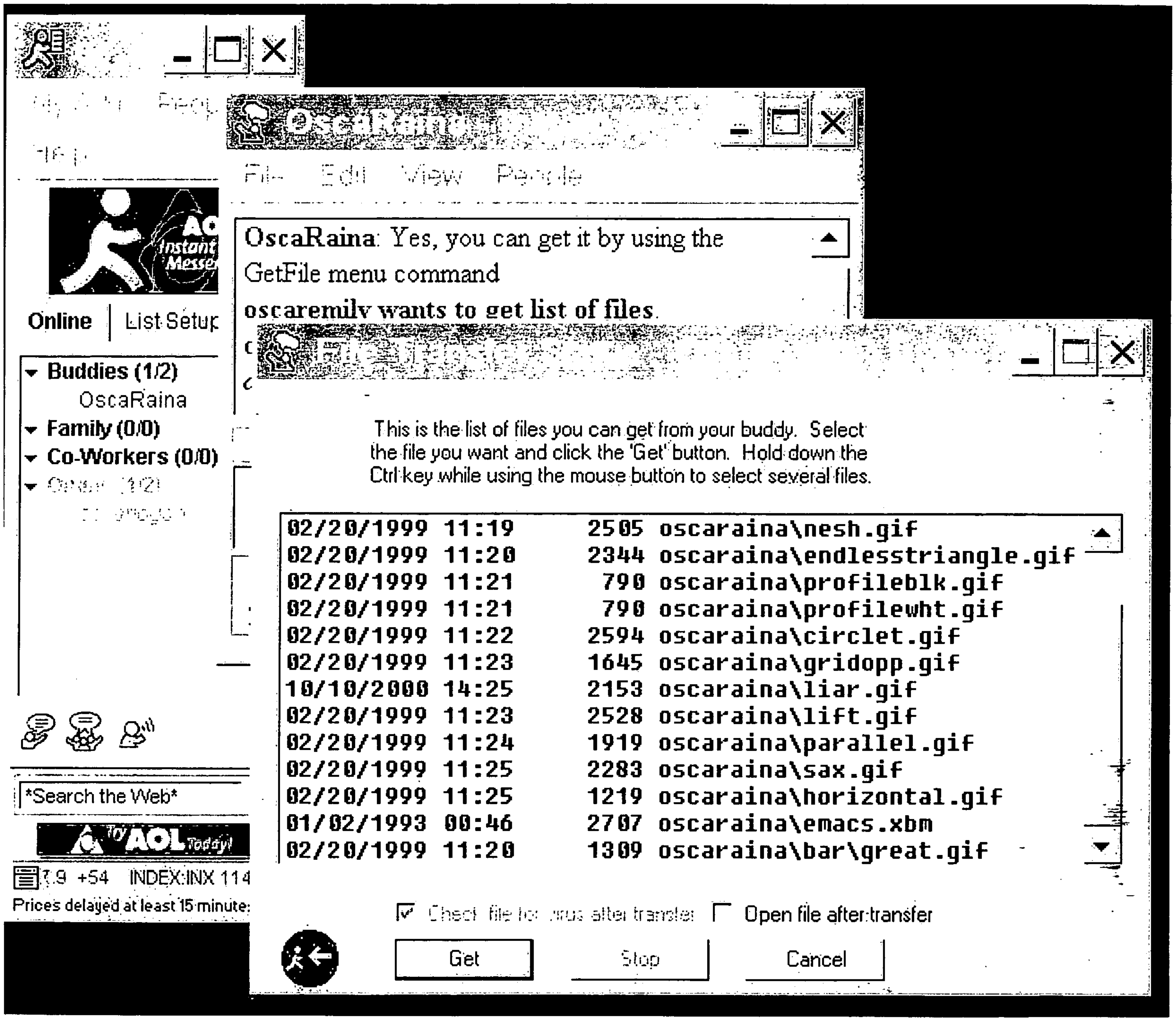
BEST AVAILABLE COPY



oscarlogan: Do you h
OscaRaina: Yes, you
menu command.

Exhibit 5

BEST AVAILABLE COPY



OscaRaina: Yes, you can get it by using the GetFile menu command

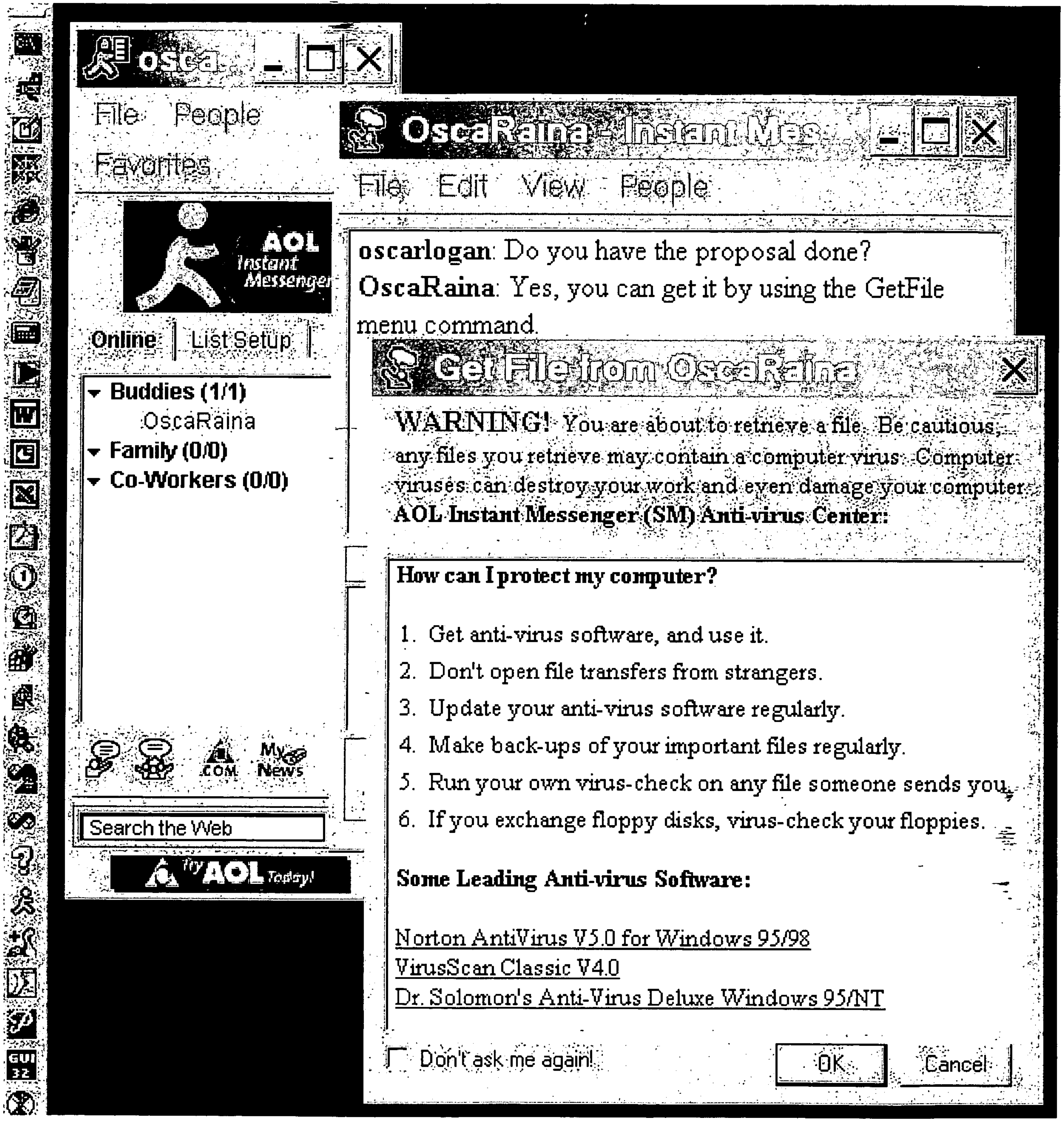
oscaremily wants to get list of files.

This is the list of files you can get from your buddy. Select the file you want and click the 'Get' button. Hold down the Ctrl key while using the mouse button to select several files.

```
02/20/1999 11:19     2505 oscaraina\nesh.gif
02/20/1999 11:20     2344 oscaraina\endlesstriangle.gif
02/20/1999 11:21      790 oscaraina\profileblk.gif
02/20/1999 11:21      790 oscaraina\profilewht.gif
02/20/1999 11:22     2594 oscaraina\circlet.gif
02/20/1999 11:23     1645 oscaraina\gridopp.gif
10/10/2000 14:25     2153 oscaraina\liar.gif
02/20/1999 11:23     2528 oscaraina\lift.gif
02/20/1999 11:24     1919 oscaraina\parallel.gif
02/20/1999 11:25     2283 oscaraina\sax.gif
02/20/1999 11:25     1219 oscaraina\horizontal.gif
01/02/1993 00:46     2707 oscaraina\emacs.xbm
02/20/1999 11:20     1309 oscaraina\bar\great.gif
```

Exhibit 6

BEST AVAILABLE COPY



oscarlogan: Do you have the proposal done?

OscaRaina: Yes, you can get it by using the GetFile menu command.

WARNING! You are about to retrieve a file. Be cautious; any files you retrieve may contain a computer virus. Computer viruses can destroy your work and even damage your computer.

AOL Instant Messenger (SM) Anti-virus Center:

How can I protect my computer?

1. Get anti-virus software, and use it.
2. Don't open file transfers from strangers.
3. Update your anti-virus software regularly.
4. Make back-ups of your important files regularly.
5. Run your own virus-check on any file someone sends you.
6. If you exchange floppy disks, virus-check your floppies.

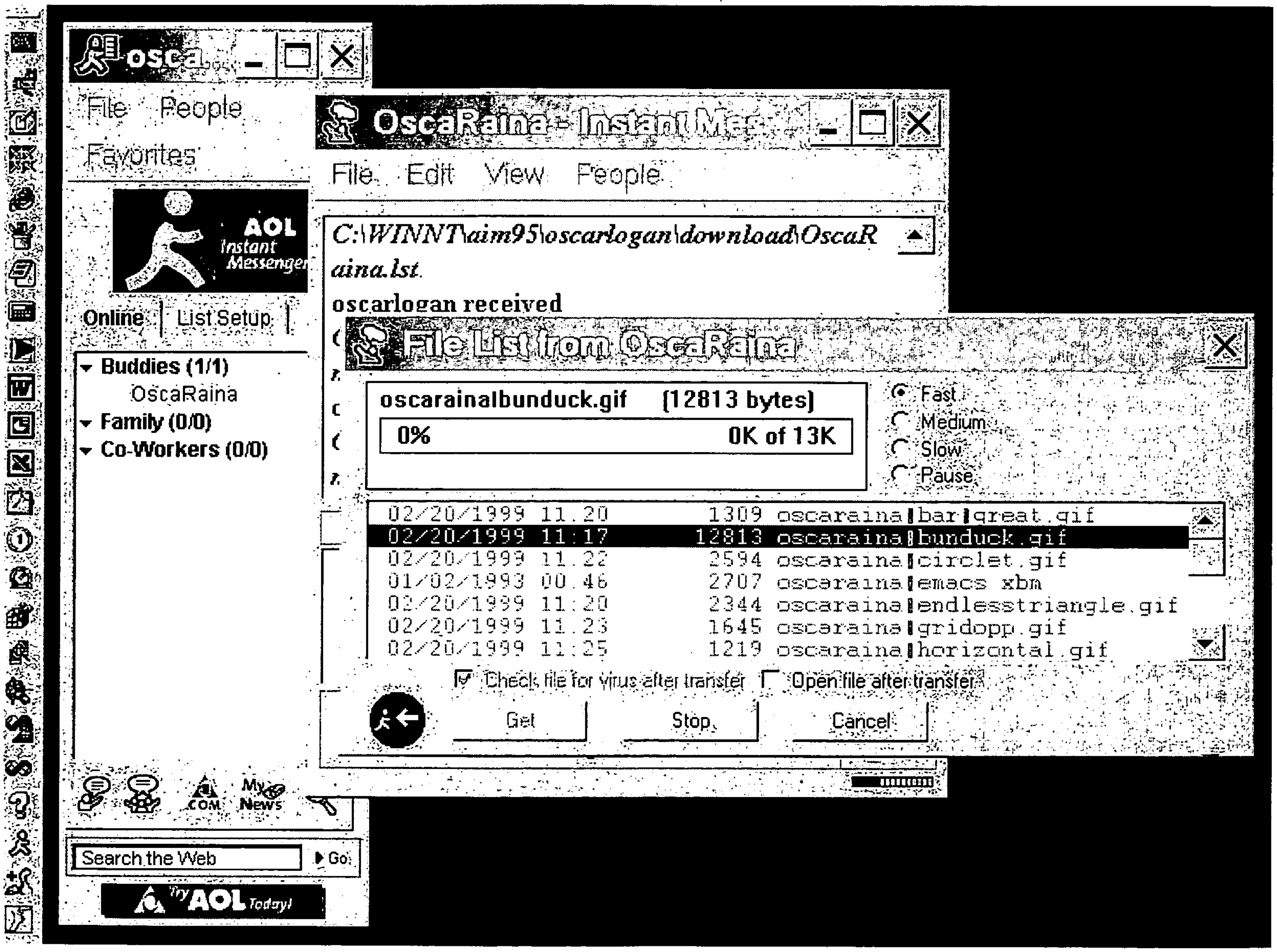
Some Leading Anti-virus Software:

Norton AntiVirus V5.0 for Windows 95/98
VirusScan Classic V4.0
Dr. Solomon's Anti-Virus Deluxe Windows 95/NT

☐ Don't ask me again! OK Cancel

Exhibit 7

BEST AVAILABLE COPY



C:\WINNT\aim95\oscarlogan\download\OscaRaina.lst.

oscarlogan received

oscarainalbunduck.gif (12813 bytes)

0% 0K of 13K

[✓ Fast] Medium Slow Pause

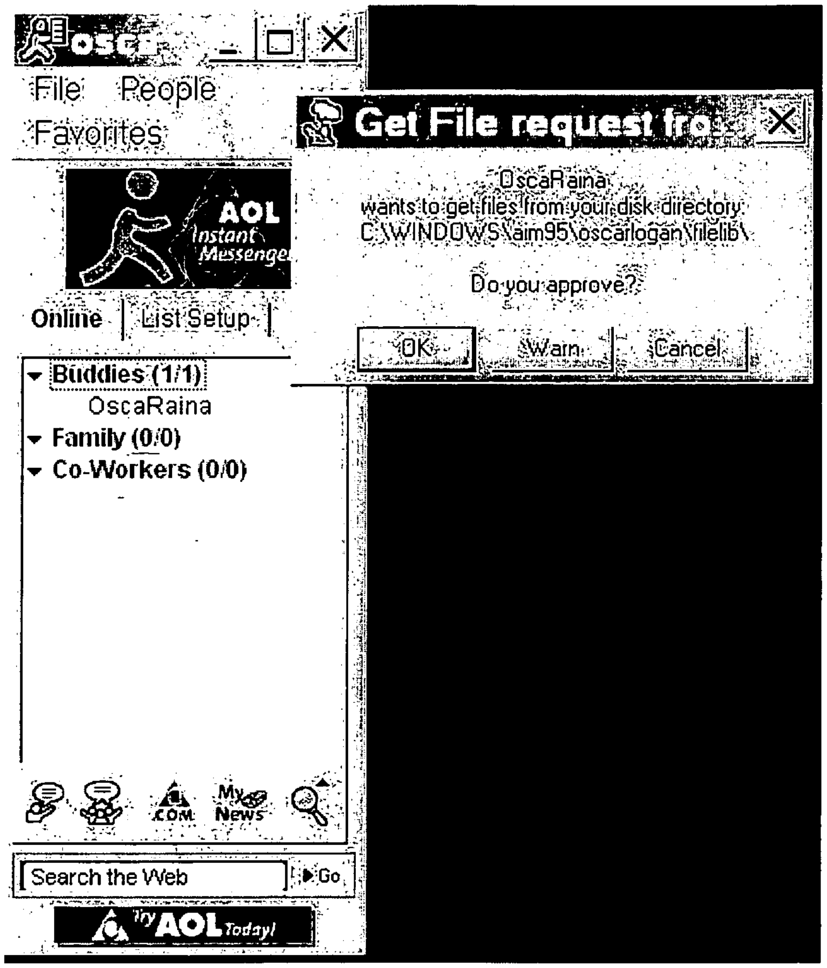
```
02/20/1999 11.20    1309 oscaraina|bar|great.gif
02/20/1999 11:17   12813 oscaraina|bunduck.gif
02/20/1999 11.22    2594 oscaraina|circlet.gif
01/02/1993 00.46    2707 oscaraina|emacs xbm
02/20/1999 11:20    2344 oscaraina|endlesstriangle.gif
02/20/1999 11.23    1645 oscaraina|gridopp.gif
02/20/1999 11:25    1219 oscaraina|horizontal.gif
```

[✓ Check file for virus after transfer] Open file after transfer

Get Stop Cancel

Exhibit 8

BEST AVAILABLE COPY



OscaRaina
wants to get files from your disk directory
C:\WINDOWS\aim95\oscarlogan\filelib\

Do you approve?

Exhibit 9